IPSJ Transactions on Computer Vision and Applications Vol.4 108–118 (Oct. 2012)

[DOI: 10.2197/ipsjtcva.4.108]

Research Paper

Optimized Codebook Construction and Assignment for Product Quantization-based Approximate Nearest Neighbor Search

Yusuke Uchida[1,a)] Koichi Takagi[1] Shigeyuki Sakazawa[1]

Received: October 31, 2011, Accepted: July 5, 2012, Released: October 19, 2012

Abstract: Nearest neighbor search (NNS) among large-scale and high-dimensional vectors has played an important role in recent large-scale multimedia search applications. This paper proposes an optimized codebook construction algorithm for approximate NNS based on product quantization. The proposed algorithm iteratively optimizes both codebooks for product quantization and an assignment table that indicates the optimal codebook in product quantization. In experiments, the proposed method is shown to achieve better accuracy in approximate NNS than the conventional method with the same memory requirement and the same computational cost. Furthermore, use of a larger number of codebooks increases the accuracy of approximate NNS at the expense of a slight increase in the memory requirement.

Keywords: approximate nearest neighbor search, high-dimensional indexing, product quantization

1. Introduction

Nearest neighbor search (NNS) in a high-dimensional space plays an important role in many computer vision algorithms and applications, where high-dimensional feature vectors such as SIFT [1] or GIST [2] are frequently used. Given a set of data points in a metric space and a query point in the same metric space, NNS is defined as the problem of identifying the data point(s) nearest to the query point. In this paper, we focus on Euclidean space NNS, which is relevant to many applications.

The kd-tree [3] is one of the best solutions for NNS in a low-dimensional space, while its effectiveness declines as dimensionality increases due to the so-called "curse of dimensionality." In order to deal with this problem, approximate approaches such as ANN [4] or LSH [5] have attracted much attention. In an approximate NNS, a search result will fail to be the exact nearest neighbor point with a probability that is characterized by the parameters of the approximate NNS algorithms. It is reported that a randomized kd-tree algorithm [6], [7] and a hierarchical k-means tree algorithm [8] perform better than ANN and LSH [8]. The randomized kd-tree algorithm constructs multiple randomized kd-trees, and these trees are explored simultaneously according to a single priority queue, which is referred to as a best-bin-first search [9]. The priority is determined by the distance between a query point and each bin boundary in the kd-trees. In Ref. [10], an improved algorithm for kd-tree construction is proposed, where a partition axis is formed by combining two or more coordinate axes instead of selecting a single coordinate axis. The hierarchical k-means tree algorithm also explores the hierarchical k-means tree [11] in a best-bin-first manner based on the distance between a query point and each branch node in the tree. The algorithm referred to as FLANN [8] optimally selects randomized kd-trees or a hierarchical k-means tree for indexing according to the given data distribution and the user's requirements, and it provides fully automated parameter selection.

For the sake of efficiency, all of the above-mentioned methods require the indexed vectors themselves to be stored in the main memory. However, this requirement is not feasible when handling large-scale datasets, e.g., when indexing millions of images [12], [13]. In order to address this issue, short code-based NNS methods are proposed: feature vectors are compressed into short codes and NNS is performed in the compressed domain. In short code-based NNS, the tradeoff between search accuracy and the size of short codes is an important performance measure in addition to the tradeoff between search accuracy and computational cost in standard approximate NNS. Because short code-based NNS methods usually perform a linear search, computational costs are almost the same among different methods depending on the size of short codes. One of the most common ways to realize a short code-based method is to utilize random projections like LSH [5]. Although the LSH algorithm has been thoroughly studied at a theoretical level, it requires short codes with a relatively large size to maintain approximate NNS accuracy. In Ref. [12], an algorithm referred to as spectral hashing (SH) has been proposed. The algorithm achieves better search accuracy with smaller codes than LSH. It formulates a binary hashing as a graph partitioning problem and solves it by assuming a uniform distribution over the data. In Ref. [14], an objective function is defined so that hash functions preserve the input distances when mapping to the Hamming space. An efficient algorithm to optimize the objective function is also proposed. In Ref. [15], a transform coding-based method is proposed. This method realizes

[1] KDDI R&D Laboratories, Inc., Fujimino, Saitama 356–8502, Japan
[a)] ys-uchida@kddilabs.jp

data-driven allocation of bits to components. It has been shown to outperform LSH and SH in approximate NNS and scene classification problems. Recently, a product quantization-based method has been proposed [13], where the distances between a query vector and quantized reference vectors can be calculated efficiently. It has been shown to outperform the methods mentioned above in terms of approximate NNS accuracy with the same short code size [13], [15]. The tradeoff between search time and accuracy is further improved by a non-exhaustive search framework.

In a non-exhaustive framework [13], a coarse quantization is first performed and the resulting residual (error) vectors are further quantized by the product quantizer. Although the residual vectors follow different distributions depending on the assigned Voronoi cells, which are defined by the coarse quantizer, the conventional method quantizes them irrespective of the assigned cells, resulting in the degradation of quantization performance. Because quantized reference vectors are used for distance calculations in the product quantization-based method, ineffective quantization leads to low search accuracy.

In this paper, in order to solve the problem of product quantization in Ref. [13], we propose a modified product quantization-based approximate NNS method, which utilizes an arbitrary number of codebooks in product quantization. Our main contribution is the development of an iterative codebook construction algorithm for product quantization. This algorithm optimizes the codebooks by iteratively executing an update step and assignment step similar to the k-means algorithm. In the update step, the codebooks are updated so that total quantization error is reduced for a fixed assignment table, which defines the assignments from residual vectors to the codebooks. In the assignment step, the assignment table is optimized for the fixed codebooks to minimize the error. The resulting optimized codebooks can reduce quantization error considerably, improving the accuracy of approximate NNS search. In experiments, the proposed method is shown to achieve better accuracy in approximate NNS compared with the conventional method with the same memory requirement and the same computational cost. Furthermore, use of a larger number of codebooks increases the accuracy of approximate NNS at the expense of a slight increase in the memory requirement.

2. Product Quantization for Nearest Neighbor Search

In this section, we briefly review the product quantization-based approximate NNS system [13]. It consists of an offline indexing step and an online search step. In the indexing step, reference vectors $\mathcal{Y}$ with a dimension of d are encoded into short codes via product quantization, and these short codes are stored in a database. In the search step, for each query vector $\mathbf{x}$ with the same dimension d, the system returns the k-nearest neighbor vectors from a k-NN search or the vectors with a distance less than a given threshold ϵ from a range search. This is accomplished by calculating the approximate distance $\dot{d}(\mathbf{x}, \mathbf{y})$ between the query vector $\mathbf{x}$ and reference vector $\mathbf{y}$ in the database. The approximate distances are efficiently calculated from the query vector $\mathbf{x}$ and the short codes in the database. The notation used in this paper is summarized in **Table 1**.

2.1 Indexing with Product Quantization

In the indexing step, a reference vector $\mathbf{y}$ is first decomposed into m d^*-dimensional subvectors $\mathbf{u}_1, \cdots, \mathbf{u}_m$:

$$\mathbf{y} = (\underbrace{y_1, \cdots, y_{d^*}}_{\mathbf{u}_1^{\mathrm{T}}}, \cdots, \underbrace{y_{d-d^*+1}, \cdots, y_d}_{\mathbf{u}_m^{\mathrm{T}}})^{\mathrm{T}}, \tag{1}$$

where $d^* = d/m$. Subsequently, these subvectors are quantized separately using m codebooks $P_1, \cdots, P_m$, which is referred to as product quantization. In this paper, a codebook used in product quantization is referred to as a PQ codebook. We assume that each PQ codebook P_l has k^* centroids $\mathbf{p}_{l,1}, \cdots, \mathbf{p}_{l,k^*} \in \mathbb{R}^{d^*}$. The l-th subvector $\mathbf{u}_l$ is quantized into $q_l(\mathbf{u}_l) = \mathbf{p}_{l,a_l}$ using l-th PQ codebook P_l, where

$$a_l = \arg\min_{1 \le a \le k^*} \|\mathbf{u}_l - \mathbf{p}_{l,a}\|^2. \tag{2}$$

As a result, $\mathbf{y}$ is quantized into $q(\mathbf{y})$:

$$q(\mathbf{y}) = (q_1(\mathbf{u}_1)^{\mathrm{T}}, \cdots, q_m(\mathbf{u}_m)^{\mathrm{T}})^{\mathrm{T}} = (\mathbf{p}_{1,a_1}^{\mathrm{T}}, \cdots, \mathbf{p}_{m,a_m}^{\mathrm{T}})^{\mathrm{T}}. \tag{3}$$

Table 1 Notations.

Symbol	Notation	Section
d	Dimensionality of feature vectors.	§2
m	The number of vector decomposition in product quantization.	§2.1
d^*	Dimensionality of subvectors, where $d^* = d/m$.	§2.1
$\mathcal{F}$	A set of training vectors with a dimension of d.	§2.1
$P_1, \cdots, P_m$	Codebooks used in the conventional product quantization (PQ codebooks), where $P_i = (\mathbf{p}_{i,1}, \cdots, \mathbf{p}_{i,k^*})$ and $\mathbf{p}_{i,a} \in \mathbb{R}^{d^*}$.	§2.1
k^*	The size of PQ codebooks (the number of centroids).	§2.1
C	A codebook used in coarse quantization (CQ codebook), where $C = (\mathbf{c}_1, \cdots, \mathbf{c}_{k'})$ and $\mathbf{c}_j \in \mathbb{R}^d$.	§2.3.1
k'	The size of CQ codebook (the number of centroids).	§2.3.1
r	The number of PQ codebooks used in the proposed method.	§3
$P_1, \cdots, P_r$	Codebooks used in the proposed product quantization (PQ codebooks), where $P_i = (\mathbf{p}_{i,1}, \cdots, \mathbf{p}_{i,k^*})$ and $\mathbf{p}_{i,a} \in \mathbb{R}^{d^*}$.	§3
$\mathcal{R}_{1,1}, \cdots, \mathcal{R}_{k',m}$	Sets of residual subvectors with a dimension of d^* used in the construction of PQ codebooks. $\mathcal{R}_{j,l}$ represents the set of l-th subvectors of the training vectors that are assigned to the j-th centroid in coarse quantization.	§3.1
T	Assignment table $T \in \mathbb{N}^{k' \times m}$. $T_{j,l}$ indicates the identifier of the PQ codebook with which l-th residual subvectors assigned to the j-th centroid in coarse quantization should be quantized.	§3.1
w	The number of inverted lists to be searched in the multiple assignment strategy.	§4.5

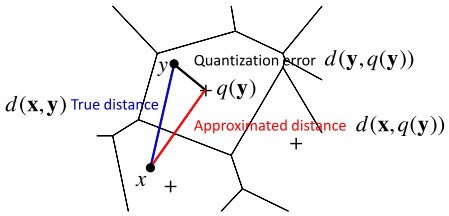


Fig. 1 Relationship among the true distance $d(\mathbf{x}, \mathbf{y})$, the approximate distance $d(\mathbf{x}, q(\mathbf{y}))$, and the quantization error $d(\mathbf{y}, q(\mathbf{y}))$.

A tuple $(a_1, \cdots, a_m)$, the short code representation of $\mathbf{y}$, is stored for search purposes. As a_l ranges from 1 to k^*, the length of the short code becomes $m\lceil \log_2 k^* \rceil$. The PQ codebooks should be created prior to indexing using a large number of training vectors $\mathcal{F}$: the l-th PQ codebook is created by clustering a set of the l-th subvectors of the training vectors $\mathcal{F}$ using the k-means algorithm.

2.2 Distance Calculation between Query Vector and Short Code

In the search step, a query vector $\mathbf{x}$ is also decomposed into m d^*-dimensional subvectors $\mathbf{u}_1, \cdots, \mathbf{u}_m$. The distance $d(\mathbf{x}, \mathbf{y})$ between the query vector $\mathbf{x}$ and a reference vector $\mathbf{y}$ is approximated by the distance $d(\mathbf{x}, q(\mathbf{y}))$ between the query vector $\mathbf{x}$ and quantized reference vector $q(\mathbf{y})$ with the short code $(a_1, \cdots, a_m)$:

$$d(\mathbf{x}, \mathbf{y}) \approx \dot{d}(\mathbf{x}, \mathbf{y}) = d(\mathbf{x}, q(\mathbf{y})) = \sqrt{\sum_{l=1}^{m} \|\mathbf{u}_l - \mathbf{p}_{l,a_l}\|^2}. \quad (4)$$

For efficiency, lookup table F is prepared when a query vector $\mathbf{x}$ is given:

$$F_{l,a} = \|\mathbf{u}_l - \mathbf{p}_{l,a}\|^2 \ (1 \le l \le m, \ 1 \le a \le k^*). \quad (5)$$

Using this lookup table, the approximate distance $\dot{d}(\mathbf{x}, \mathbf{y})$ is calculated:

$$\dot{d}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{l=1}^{m} F_{l,a_l}}. \quad (6)$$

Figure 1 illustrates the relationship among the true distance $d(\mathbf{x}, \mathbf{y})$, the approximate distance $d(\mathbf{x}, q(\mathbf{y}))$, and the quantization error $d(\mathbf{y}, q(\mathbf{y}))$. The triangular inequality gives

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &\le d(\mathbf{x}, q(\mathbf{y})) + d(\mathbf{y}, q(\mathbf{y})), \\ d(\mathbf{x}, q(\mathbf{y})) &\le d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, q(\mathbf{y})). \end{aligned} \quad (7)$$

The equation can be modified as

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) - d(\mathbf{x}, q(\mathbf{y})) &\le d(\mathbf{y}, q(\mathbf{y})), \\ -d(\mathbf{y}, q(\mathbf{y})) &\le d(\mathbf{x}, \mathbf{y}) - d(\mathbf{x}, q(\mathbf{y})). \end{aligned} \quad (8)$$

Finally, we obtain:

$$|d(\mathbf{x}, \mathbf{y}) - d(\mathbf{x}, q(\mathbf{y}))| \le d(\mathbf{y}, q(\mathbf{y})). \quad (9)$$

Here, $|d(\mathbf{x}, \mathbf{y}) - d(\mathbf{x}, q(\mathbf{y}))|$ corresponds to the error in distance approximation, and is bounded by the quantization error $d(\mathbf{y}, q(\mathbf{y}))$.

2.3 Integration with Inverted Index

Although approximate nearest neighbor search with product quantizers is fast, the search is exhaustive and there is room for improvement. The product quantization-based scheme can be integrated with an inverted index to avoid exhaustive searches and thus further increase efficiency. The non-exhaustive framework is referred to as an inverted file with asymmetric distance computation (IVFADC) in Ref. [13].

2.3.1 Indexing in IVFADC

In the indexing step in the IVFADC framework, a reference vector $\mathbf{y}$ is first quantized with a coarse quantizer. We refer to the codebook used in coarse quantization as the CQ codebook. The reference vector $\mathbf{y}$ is quantized into $q^c(\mathbf{y}) = \mathbf{c}_{\hat{j}}$ using the CQ codebook C with k' centroids $\mathbf{c}_1, \cdots, \mathbf{c}_{k'} \in \mathbb{R}^d$, where

$$\hat{j} = \arg\min_{1 \le j \le k'} \|\mathbf{y} - \mathbf{c}_j\|^2. \quad (10)$$

Subsequently, the residual vector $\mathbf{r}_{\hat{j}}$ from the corresponding centroid $\mathbf{c}_{\hat{j}}$ is calculated as

$$\mathbf{r}_{\hat{j}} = \mathbf{y} - \mathbf{c}_{\hat{j}}. \quad (11)$$

Then, in order to reduce quantization error in the coarse quantizer, the residual vector $\mathbf{r}_{\hat{j}}$ is quantized via product quantization in the same manner as described in Section 2.1. The residual vector $\mathbf{r}_{\hat{j}}$ is divided into m subvectors $\mathbf{r}_{\hat{j},1}, \cdots, \mathbf{r}_{\hat{j},m}$. The residual subvector $\mathbf{r}_{\hat{j},l}$ is quantized into $q_l(\mathbf{r}_{\hat{j},l}) = \mathbf{p}_{l,a_l}$ using the PQ codebook P_l, where

$$a_l = \arg\min_{1 \le a \le k^*} \|\mathbf{r}_{\hat{j},l} - \mathbf{p}_{l,a}\|^2. \quad (12)$$

Finally, the short code $(a_1, \cdots, a_m)$ is stored in the $\hat{j}$-th list of the inverted index with the vector identifier.

2.3.2 Distance Calculation in IVFADC

In the search step in the IVFADC framework, a query vector $\mathbf{x}$ is first quantized using the CQ codebook, and the residual vector $\mathbf{r}_{\hat{j}}$ from the corresponding centroid is calculated. Subsequently, the approximate distances between the residual vector $\mathbf{r}_{\hat{j}}$ and the short codes in the index are calculated. These distances correspond to the approximate distances between the query vector and the reference vectors. In contrast to the exhaustive search described in Section 2.2, only the short codes in the $\hat{j}$-th list of the inverted index are concerned, where

$$\hat{j} = \arg\min_{1 \le j \le k'} \|\mathbf{x} - \mathbf{c}_j\|^2. \quad (13)$$

Compared with the exhaustive search, IVFADC is shown to achieve a better tradeoff between approximate NNS accuracy and search speed [13].

2.3.3 Codebook Construction for IVFADC

In the case of IVFADC, the CQ codebook and PQ codebooks should be created prior to indexing. The CQ codebook is constructed by clustering a large set of the training vectors $\mathcal{F}$ using the k-means algorithm to obtain k' centroids $\mathbf{c}_1, \cdots, \mathbf{c}_{k'} \in \mathbb{R}^d$. Then, by subtracting corresponding centroid vectors from training vectors, a set of residual vectors is created. Subsequently, the residual vectors are divided into m residual subvectors. Finally, for each l, the l-th PQ codebook is constructed by clustering a set of l-th residual subvectors of the residual vectors using the k-means algorithm.

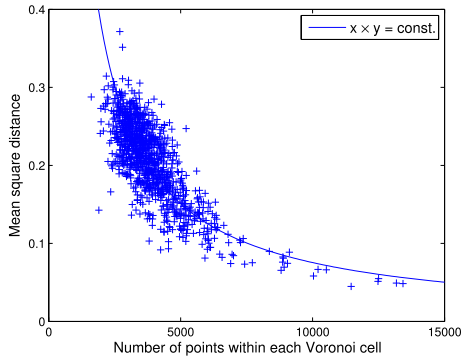


Fig. 2 The number of feature vectors assigned to a centroid and the mean square distance between the feature vectors and the centroid are plotted. The CQ codebook with a size of 1,024 is created from 4M SIFT feature vectors with the k-means algorithm.

2.4 Ineffectiveness in Product Quantization

In Ref. [13], residual subvectors from the same position in vector decomposition are quantized with the same PQ codebook irrespective of the cells into which corresponding residual vectors are quantized in coarse quantization. Because these cells have different residual vector distributions, the quantization of residual subvectors from different distributions with the same PQ codebook is ineffective in terms of reducing quantization error, which results in the degradation of approximate NNS accuracy. **Figure 2** shows the relationship between the number of feature vectors assigned to a centroid and the mean square distance between the feature vectors and the centroid. Since the mean square distance reflects the size of a cell, Fig. 2 indicates that there are marked differences in the size of the Voronoi cells, and it implies that the distributions of residual vectors are also quite different depending on the cells. To handle this diversity, an identical PQ codebook for each cell and for each position of residual subvectors may be used for product quantization. However, the memory requirements for the PQ codebooks become very large relative to the number of centroids k' in the coarse quantization. This requirement would be intractable in some situations where large k' (e.g., from 10K to 1M in Ref. [6]) is often used. For example, assuming that the number of centroids k' of the CQ codebook is 100K, the number of centroids k^* of each PQ codebook is 256, and 128-dimensional SIFT features are indexed; this means that approximately $100\text{K} \times 256 \times 128 \simeq 3$ Gbytes of memory is required to store only the PQ codebooks. Furthermore, to generalize the PQ codebooks to non-training vectors, k' times as many training vectors as in the case of creating a single PQ codebook for each residual subvector position are required.

3. Proposed Approach

As explained in Section 2, residual subvectors have different distributions depending on the assigned centroids in coarse quantization. In this paper, we propose the use of an arbitrary number r ($1 \leq r \leq k' \times m$) of PQ codebooks $P_1, \cdots, P_r$ in product quantization, where residual subvectors from similar distributions share the same PQ codebook. In the conventional method, a residual subvector is quantized by a PQ codebook that is specified only by the position l of the residual subvector in vector decomposition, while the proposed method specifies a PQ codebook not only by the position l of the residual subvector but also by the identifier j of the centroid into which the corresponding reference vector is quantized in coarse quantization. **Figure 3** shows an intuitive illustration of the proposed approach. The proposed algorithm creates semi-optimal PQ codebooks to minimize the expected mean square error (MSE) in product quantization for a given r. Because the error in product quantization corresponds to the upper bound of error in distance approximation as shown in Section 2.2, it is very important to reduce the error to improve the accuracy of product quantization-based approximate NNS. In this section, an algorithm to create an arbitrary number r of optimized PQ codebooks to minimize RMSE is proposed. In the algorithm, an assignment table T is also created. The table specifies the PQ codebook to be used in the quantization of a residual subvector depending on the position of the residual subvector and the centroid into which the corresponding residual vector is quantized in coarse quantization. Indexing and search algorithms using optimized PQ codebooks are also described.

3.1 Multiple Codebook Construction

In this section, the codebook construction algorithm is described. It includes the construction of CQ codebook C for coarse quantization, PQ codebooks $P_1, \cdots, P_r$ for product quantization, and an assignment table T. We first construct a CQ codebook C by clustering the set of training vectors $\mathcal{F}$ with the k-means algorithm. The residual vectors of the training vectors from their corresponding centroids in C are calculated. Subsequently, all the residual vectors are divided into m residual subvectors with a dimension of d^*. Let $\mathcal{R}_{j,l}$ denote the set of l-th residual subvectors that are assigned to the j-th centroid in coarse quantization. Our objective is to create optimized PQ codebooks $P_1, \cdots, P_r$ and an assignment table T that minimize the expected MSE in product quantization. Here, $T_{j,l}$ indicates the identifier of a PQ codebook with which l-th residual subvectors assigned to the j-th centroid in coarse quantization should be quantized. The objective is formulated as

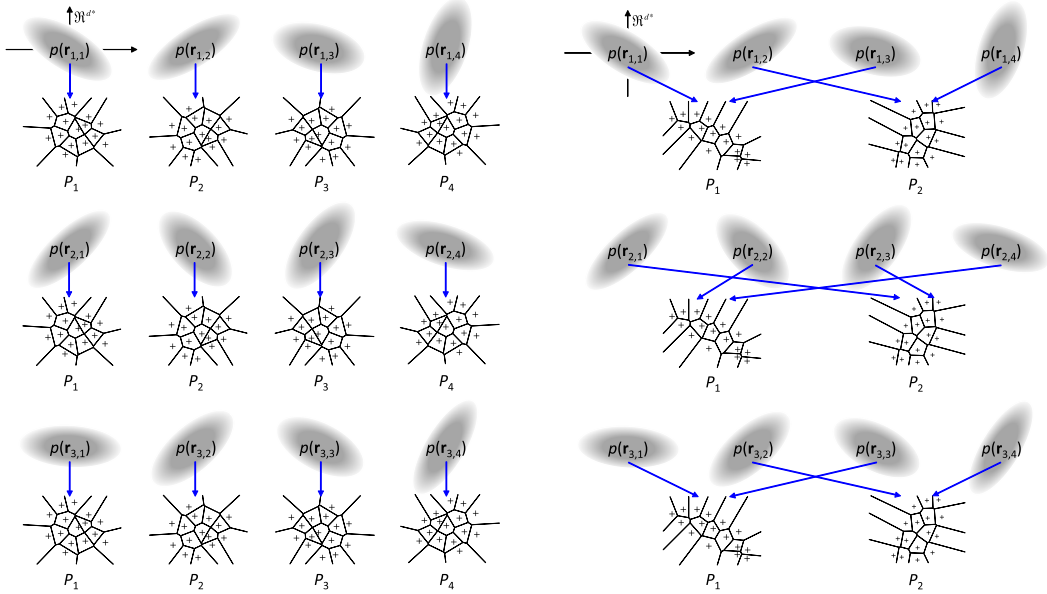
$$\text{minimize: } \sum_{j=1}^{k'} \sum_{l=1}^{m} e(\mathcal{R}_{j,l}, P_{T_{j,l}}), \tag{14}$$

where $e(\mathcal{R}_{j,l}, P_i)$ represents the sum of the squares of errors in quantizing a set of residual subvectors $\mathcal{R}_{j,l}$ with PQ codebook P_i:

$$e(\mathcal{R}_{j,l}, P_i) = \sum_{1 \leq b \leq |\mathcal{R}_{j,l}|} \min_{1 \leq a \leq k^*} \|\mathbf{r}_{j,l,b} - \mathbf{p}_{i,a}\|^2. \tag{15}$$

In the above equation, $\mathbf{r}_{j,l,b}$ represents the b-th training subvector in $\mathcal{R}_{j,l}$. This optimization problem requires that the assignment table T and the PQ codebooks $P_1, \cdots, P_r$ should be optimized simultaneously. Because it is an NP-hard problem, we propose an approximate algorithm which iteratively optimizes (1) the assignment table T for the fixed PQ codebooks $P_1, \cdots, P_r$ and (2) the PQ codebooks for the fixed assignment table T.

The proposed codebook construction procedure is summarized in **Algorithm 1**. The algorithm consists of an initialization step (Line 1), an update step (Lines 3–5), and an assignment step (Lines 6–22). In the initialization step, the assignment table T and initial labels $\mathbf{s}_{j,l} = (s_{j,l,1}, \cdots, s_{j,l,|\mathcal{R}_{j,l}|})$ are initialized. The initialization of T is described in Section 3.2 in detail. The label $s_{j,l,b}$



(a) Product quantization in the conventional method. (b) Product quantization in the proposed method.

Fig. 3 Intuitive illustration of the proposed approach for $k' = 3, m = 4, r = 2$. Each ellipse illustrates the probability density function $p(\mathbf{r}_{j,l})$ defined in the feature space $\mathbb{R}^{d^*}$, where $\mathbf{r}_{j,l}$ is the l-th residual subvector of a reference vector that is assigned to the j-th centroid in coarse quantization. Darker area corresponds to higher density. Probability distribution $p(\mathbf{r}_{j,l})$ is different depending on both j and l. PQ codebooks are represented by Voronoi diagrams. Each blue arrow points to the PQ codebook that is used to quantize the residual subvector $\mathbf{r}_{j,l}$. (a) In the conventional scheme, each residual subvector $\mathbf{r}_{j,l}$ is assigned to the PQ codebook according to only l; residual subvectors that follow quite different distrubutions may be quantized with the same PQ codebook, resulting in large quantization error. (b) In the proposed scheme, each residual subvector $\mathbf{r}_{j,l}$ is optimally assigned to the PQ codebook according to not only l but j; residual subvectors that follow similar distributions tend to be quantized with the same PQ codebook adapted to the distributions, which reduces quantization error.

Algorithm 1 PQ codebook construction

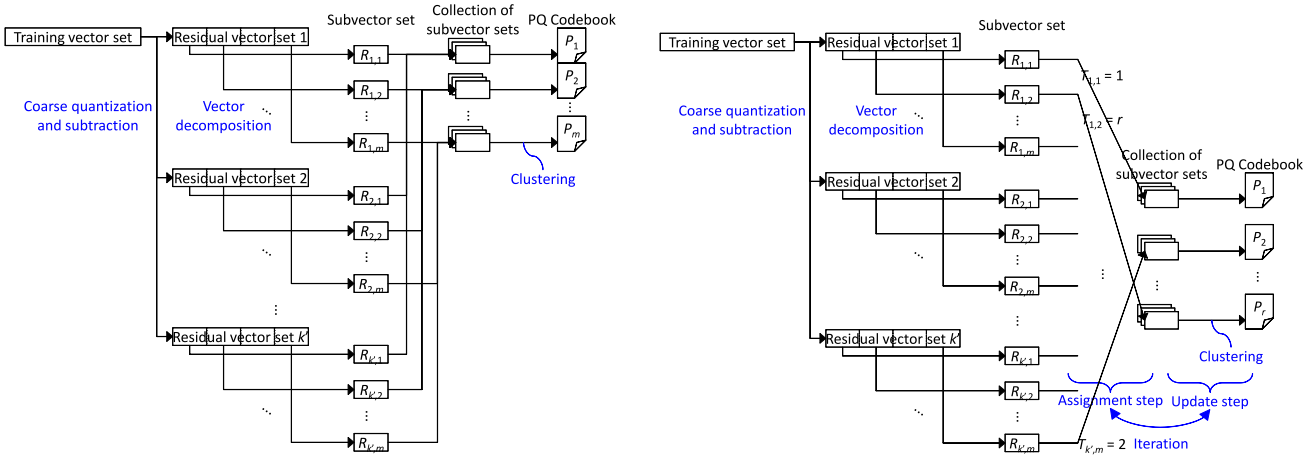
Require: $\mathcal{R}_{j,l}$ $(1 \le j \le k', 1 \le l \le m)$, r
Ensure: $P_1, \cdots, P_r$, T
1: initialize T, $\mathbf{s}_{j,l}$ $(1 \le j \le k', 1 \le l \le m)$
2: **repeat**
3: **for** $i = 1$ **to** r **do**
4: update P_i by clustering $\bigcup_{T_{j,l}=i} \mathcal{R}_{j,l}$ using initial labels $\mathbf{s}_{j,l}$ for $\mathcal{R}_{j,l}$
5: **end for**
6: **for** $j = 1$ **to** k' **do**
7: **for** $l = 1$ **to** m **do**
8: $e \leftarrow \infty$
9: **for** $i = 1$ **to** r **do**
10: $\hat{e} \leftarrow 0$
11: **for** $b = 1$ **to** $|\mathcal{R}_{j,l}|$ **do**
12: $\hat{e} \leftarrow \hat{e} + \min_{1 \le a \le k^*} \|\mathbf{r}_{j,l,b} - \mathbf{p}_{i,a}\|^2$
13: $\hat{s}_{j,l,b} \leftarrow \mathrm{argmin}_{1 \le a \le k^*} \|\mathbf{r}_{j,l,b} - \mathbf{p}_{i,a}\|^2$
14: **end for**
15: **if** $e > \hat{e}$ **then**
16: $e \leftarrow \hat{e}$
17: $T_{j,l} \leftarrow i$
18: $\mathbf{s}_{j,l} \leftarrow \hat{\mathbf{s}}_{j,l}$
19: **end if**
20: **end for**
21: **end for**
22: **end for**
23: **until** a fixed number of iterations are performed

is used as the initial label of the b-th training subvector $\mathbf{r}_{j,l,b}$ in the k-means clustering in the update step. The label $s_{j,l,b}$ is initialized by a random value ranging from 1 to k^* in the initialization step, and is updated with the identifier of the centroid in P_i closest to $\mathbf{r}_{j,l,b}$ in the assignment step. The update step and assignment step are iterated for a fixed number of times.

Update step. In the update step, Eq. (14) is minimized by updating the PQ codebooks $P_1, \cdots, P_r$ for the fixed assignment table T. For each i, the i-th PQ codebook P_i is updated using the residual subvectors assigned to P_i. As the assignment is defined by the assignment table T, the update step is simply achieved by clustering the residual subvectors $\bigcup_{T_{j,l}=i} \mathcal{R}_{j,l}$ to obtain k^* centroids $\mathbf{p}_{i,1}, \cdots, \mathbf{p}_{i,k^*}$. In the clustering, the initial label $s_{j,l,b}$ is used for the b-th training subvector $\mathbf{r}_{j,l,b}$ in $\mathcal{R}_{j,l}$ to accelerate and stabilize the clustering instead of starting with random labels. As the labels $\mathbf{s}_{j,l} = (s_{j,l,1}, \cdots, s_{j,l,|\mathcal{R}_{j,l}|})$ for $\mathcal{R}_{j,l}$ are obtained in the assignment step, random labels are used only in the first update step.

Assignment step. In the assignment step, Eq. (14) is minimized by updating the assignment table T for the fixed PQ codebooks $P_1, \cdots, P_r$. For each $\mathcal{R}_{j,l}$, the quantization error $\hat{e}$ in quantizing $\mathcal{R}_{j,l}$ with the PQ codebook P_i is calculated for all i (lines 9–20). The identifier of the PQ codebook that minimizes the quantization error is stored in $T_{j,l}$ (line 17). The identifier of the centroid in P_i closest to $\mathbf{r}_{j,l,b}$ is also stored in $s_{j,l,b}$ (line 13 and 18) for the update step.

The difference between the proposed method and the conventional method for the construction of PQ codebooks is summarized in **Fig. 4**. The proposed method becomes identical to the conventional method when we set $r = m$ and $T_{j,l} = l$. CQ code-



(a) PQ codebook construction procedure in the conventional method.

(b) PQ codebook construction procedure in the proposed method.

Fig. 4 The difference between the conventional method and the proposed method for the construction of PQ codebooks. (a) In the conventional method, the l-th PQ codebook is constructed by clustering a collection of residual subvectors $\mathcal{R}_{1,l}, \cdots, \mathcal{R}_{k',l}$ irrespective of the centroids to which the corresponding training vectors are assigned in coarse quantization. (b) In the proposed method, the i-th PQ codebook is constructed by clustering a collection of residual subvectors $\mathcal{R}_{j,l}$ such that $T_{j,l} = i$. The assignment table T and the PQ codebooks $P_1, \cdots, P_r$ are iteratively optimized in the assignment step and the update step, respectively.

book C, PQ codebooks $P_1, \cdots, P_r$ and the table T created in this procedure are used in both the indexing and search procedures.

Computational cost. The computational cost required by the proposed algorithm is examined. The update step involves performing k-means clustering. It costs $O(n_{\text{ave}} \cdot n_{\text{itr}} \cdot k^* \cdot d^*)$ time to update each PQ codebook, where n_{ave} denotes the average number of residual subvectors assigned to each PQ codebook, and n_{itr} denotes the number of iterations in k-means clustering. Let n_{all} denote the number of all training subvectors; it costs $O(r \cdot n_{\text{ave}} \cdot n_{\text{itr}} \cdot k^* \cdot d^*) = O(n_{\text{all}} \cdot n_{\text{itr}} \cdot k^* \cdot d^*)$ time to update all PQ codebooks to be computed. The assignment step requires the distances between all training subvectors $\bigcup_{j,l} \mathcal{R}_{j,l}$ and all centroids of the PQ codebooks $P_1 \cdots, P_r$. It costs $O(n_{\text{all}} \cdot r \cdot k^* \cdot d^*)$ time. While n_{itr} is fixed at a relatively small value in this paper ($n_{\text{itr}} = 5$), r ranges from 1 to $k' \times m$. Therefore, the computational cost of our PQ codebook construction algorithm is $O(n_{\text{all}} \cdot r \cdot k^* \cdot d^*)$ in many cases.

3.2 Initialization in PQ Codebook Construction

In this section, we introduce two algorithms to initialize the assignment table T for the construction of the PQ codebooks. The first is a very simple approach: assign random labels to table T in the same manner as that used in the k-means algorithm. The other is a k-means++-like initialization algorithm inspired by the k-means++ algorithm [16], where initial centroids are chosen iteratively until the predefined number of centroids is selected; the next centroid is chosen from all samples according to a probability proportional to the squared distances to the nearest centroids that have been already chosen. The k-means++ algorithm improves both the speed and the accuracy of k-means by carefully choosing initial seeds. In our case, samples and centroids correspond to sets of residual subvectors and PQ codebooks, respectively. The squared distance between a sample and a centroid corresponds to the sum of squares of errors in the quantization of a set of residual subvectors using a PQ codebook.

Algorithm 2 Initialization in the PQ codebook construction

Require: $\mathcal{R}_{j,l}$ $(1 \le j \le k', 1 \le l \le m)$, r
Ensure: T

```
1: randomly select R̂_{j,l} and create PQ codebook P_1 by clustering it
2: for j = 1 to k' do
3:     for l = 1 to m do
4:         e_{j,l} ← e(R_{j,l}, P_1)
5:         T_{j,l} ← 1
6:     end for
7: end for
8: for i = 2 to r do
9:     select R̂_{j,l} with the probability e_{j,l} / Σ_{j=1}^{k'} Σ_{l=1}^{m} e_{j,l} and create PQ codebook P_i by clustering it
10:    for j = 1 to k' do
11:        for l = 1 to m do
12:            if e_{j,l} > e(R_{j,l}, P_i) then
13:                e_{j,l} ← e(R_{j,l}, P_i)
14:                T_{j,l} ← i
15:            end if
16:        end for
17:    end for
18: end for
```

The k-means++-like initialization algorithm is summarized in **Algorithm 2**. In the algorithm, the PQ codebooks $P_1, \cdots, P_r$ are created in order, and the assignment table T is updated accordingly. First, P_1 is created by clustering randomly selected $\hat{\mathcal{R}}_{j,l}$ (line 1). All residual subvectors are assigned to P_1 and the quantization error $e_{j,l}$ in quantizing $\mathcal{R}_{j,l}$ with the PQ codebook P_1 is calculated (line 4–5). Then, $P_2, \cdots, P_r$ are created recursively. For each i, P_i is created by clustering $\hat{\mathcal{R}}_{j,l}$, where $\hat{\mathcal{R}}_{j,l}$ is selected with a probability proportional to the quantization error $e_{j,l}$. The quantization error $e_{j,l}$ corresponds to the minimum quantization error in quantizing $\mathcal{R}_{j,l}$ achieved using one of the previously created PQ codebooks $P_1, \cdots, P_{i-1}$ (line 9). For each $\mathcal{R}_{j,l}$, the assignment table and the quantization error is updated if the new PQ codebook P_i produces a lower quantization error than the current best PQ

codebook (line 12–15). After the PQ codebooks $P_1, \cdots, P_r$ are created, the initial assignment table T is obtained.

3.3 Indexing Using Optimized PQ Codebooks

The indexing step is almost the same as in the conventional method described in Section 2.3.1 except that, in our scheme, the PQ codebook used in residual subvector quantization is identified by the assignment table T. The reference vector $\mathbf{y}$ is quantized into $q^c(\mathbf{y}) = \mathbf{c}_{\hat{j}}$ using CQ codebook C:

$$\hat{j} = \arg\min_{1 \le j \le k'} \|\mathbf{y} - \mathbf{c}_j\|^2. \tag{16}$$

The residual vector $\mathbf{r}_{\hat{j}}$ from the corresponding centroid is calculated thus:

$$\mathbf{r}_{\hat{j}} = \mathbf{y} - \mathbf{c}_{\hat{j}}. \tag{17}$$

Residual vector $\mathbf{r}_{\hat{j}}$ is then divided into m subvectors $\mathbf{r}_{\hat{j},1}, \cdots, \mathbf{r}_{\hat{j},m}$. The PQ codebook $P_{\hat{i}}$ used for the l-th residual subvector is identified by the assignment table T as $\hat{i} = T_{\hat{j},l}$. Then residual subvector $\mathbf{r}_{\hat{j},l}$ is quantized into $q_{\hat{i}}(\mathbf{r}_{\hat{j},l}) = \mathbf{p}_{\hat{i},a_l}$, where

$$a_l = \arg\min_{1 \le a \le k^*} \|\mathbf{r}_{\hat{j},l} - \mathbf{p}_{\hat{i},a}\|^2. \tag{18}$$

Finally, the short code $(a_1, \cdots, a_m)$ is stored in the $\hat{j}$-th list of the inverted index with the vector identifier.

3.4 Distance Calculation Using Optimized PQ Codebooks

In the search step, query vector $\mathbf{x}$ is first quantized using the CQ codebook C:

$$\hat{j} = \arg\min_{1 \le j \le k'} \|\mathbf{x} - \mathbf{c}_j\|^2. \tag{19}$$

The residual vector $\mathbf{r}_{\hat{j}}$ of $\mathbf{x}$ is calculated and divided into m subvectors $\mathbf{r}_{\hat{j},1}, \cdots, \mathbf{r}_{\hat{j},m}$ in the same manner as in the indexing step. Then, the distance table F is created for subsequent distance calculations. Note that short codes in the $\hat{j}$-th list in the inverted index are created using the PQ codebooks $P_{T_{\hat{j},1}}, \cdots, P_{T_{\hat{j},m}}$, instead of $P_1, \cdots, P_m$.

$$F_{l,a} = \|\mathbf{r}_{\hat{j},l} - \mathbf{p}_{T_{\hat{j},l},a}\|^2 \quad (1 \le l \le m, 1 \le a \le k^*). \tag{20}$$

There is no additional computational cost caused by the use of a larger-than-m number r of PQ codebooks because the size of table F is still the same as in the conventional method ($m \times k^*$). Finally, the approximate distances between the query vector and the reference vectors in the $\hat{j}$-th list of the inverted index are calculated. The approximate distance between the query vector $\mathbf{x}$ and the reference vector $\mathbf{y}$ with code $(a_1, \cdots, a_m)$ is efficiently calculated using the lookup table F:

$$\dot{d}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{l=1}^{m} F_{l,a_l}} \tag{21}$$

This step also requires the same computational cost as that in the conventional method.

4. Experimental Results

In this section, the datasets used for the evaluation are briefly presented. Then, we evaluate the proposed method and the conventional method in terms of quantization error and accuracy in NNS in detail.

Table 2 Summary of the SIFT and GIST data sets.

	(HesAff-)SIFT	Grid-SIFT	GIST
dimensionality	128	128	384
# of training vectors	4,000,000	4,000,000	4,000,000
# of reference vectors	1,000,000	-	1,000,000
# of query vectors	100,000	-	100,000

4.1 Datasets

In our experiments, datasets with local SIFT descriptors [1] and global GIST descriptors [2], which are frequently used in the area of image retrieval and recognition, are used to evaluate the accuracy of approximate NNS. For each of the two descriptors, three types of vectors are required to evaluate the accuracy of approximate NNS [13]: training vectors, reference vectors, and query vectors. Training vectors are used to construct all of the CQ and PQ codebooks.

In the case of the SIFT descriptor, ANN_SIFT1M descriptors [*1] are used as reference vectors. Although the dataset also includes 100K training vectors and 10K query vectors, the number of training vectors is insufficient for the proposed method, as shown in Section 4.4. To deal with this problem, 4M training vectors and 100K query vectors are extracted from Flickr60K descriptors, which form a part of the INRIA Holidays dataset [*2]. We refer the above-mentioned SIFT descriptor as HesAff-SIFT because it is extracted Hessian-Affine region [17], while another type of SIFT descriptor called Grid-SIFT is also introduced in Section 4.3. As the Grid-SIFT descriptor is used only in Section 4.3, we simply refer to the HesAff-SIFT descriptor as SIFT in the other sections.

For the GIST descriptor, 4M training vectors, 1M reference vectors, and 100K query vectors are extracted from the 80 Million Tiny Images dataset [*3], as provided in Ref. [18]. Since this includes not only 32×32 color images but also precomputed 384-dimensional GIST feature vectors, these precomputed vectors are used in this paper. The SIFT and GIST data sets used in the experiments are summarized in **Table 2**.

The following experiments were performed on a machine with a Core i7 970 CPU and 24 GB of main memory. Parameters used in the experiments follow the parameters shown to be appropriate in Ref. [13]: $k' = 1{,}024$, $m = 8$, $k^* = 256$ for the SIFT vectors, and $k' = 1{,}024$, $m = 8, 24$, $k^* = 256$ for the GIST vectors. Coarse and product quantization were performed by exhaustive search with SIMD instructions. Processing times were measured using a program with a single thread.

4.2 The Impact of Initialization Methods

In this section, we evaluate the two initialization methods described in Section 3.2 using the root mean square error (RMSE) measure, the random assignment method (`base`) and the k-means++-like initialization method (`k-means++`). A CQ codebook with a size of 1,024 created from the 4M training SIFT vectors is used for coarse quantization. This CQ codebook is also used in the following experiments. The number r of PQ codebooks is fixed at 64, and 4M training vectors are used to create PQ codebooks. **Figure 5** shows RMSE and elapsed time at the

[*1] http://corpus-texmex.irisa.fr/
[*2] http://lear.inrialpes.fr/~jegou/data.php
[*3] http://horatio.cs.nyu.edu/mit/tiny/data/index.html

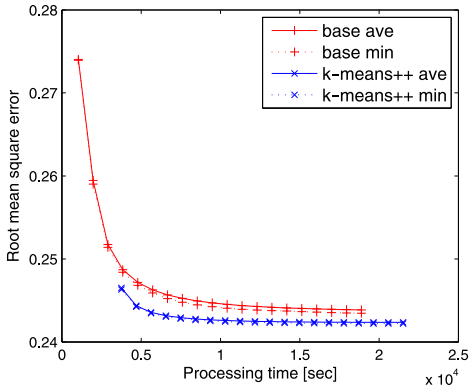


Fig. 5 Comparison of initialization methods. RMSE at the end of each iteration and elapsed time at that point are shown for up to 20 iterations. The average and minimum RMSEs over 10 runs are plotted.

end of each iteration in the construction of PQ codebooks. We can see that although the k-means++-like initialization method requires additional processing time at the first iteration, it provides a better tradeoff between computational cost and RMSE. PQ codebooks with smaller RMSE provide more accurate distance approximation, resulting in better accuracy of approximate NNS. In the following experiments, all PQ codebooks are created with the k-means++-like initialization, and the number of iterations is set to 10.

4.3 Adjusting PQ Codebooks and Assignment Table

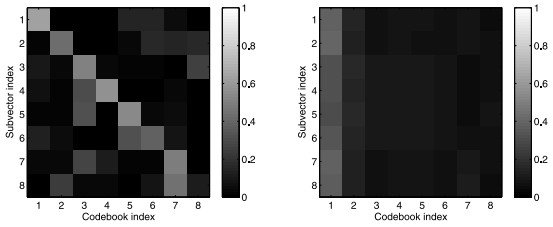
The proposed codebook construction algorithm adjusts both PQ codebooks and an assignment table to incoming vectors by iterative optimization. In this section, we explore the optimization capacity using two types of SIFT vectors with different characteristics. One is a SIFT vector describing local invariant regions detected by the Hessian-Affine detector (HesAff-SIFT). The other one is a SIFT vector describing a regular grid [19] in an image (Grid-SIFT). HesAff-SIFT represents feature vectors whose subvectors follow different distributions depending on the positions of subvectors, while Grid-SIFT represents feature vectors whose subvectors follow the same distribution independent of the positions of subvectors. In typical SIFT implementations, a target patch to be described is divided into 4×4 blocks and an 8-dimensional subvector is extracted from each of the blocks, resulting in a 128-dimensional SIFT vector. In the case of HesAff-SIFT, subvectors of different blocks follow different distributions due mainly to two reasons; (1) a patch to be described includes a blob-like area: some blocks corresponds to the blob-like area and the others correspond to non blob-like area, (2) the contribution of each pixel in a patch to feature vectors is weighted by a Gaussian function of the distance from the center of the patch (pixels far from the center have less impact on feature vectors). Although many implementations have also adopted a Gaussian weighting function in describing Grid-SIFT feature vectors, in order to compare two different types of feature vectors, we do not use Gaussian weighting for Grid-SIFT: the Grid-SIFT vectors used in the experiment are extracted from randomly downloaded Flickr images using a modified version of the VLFeat library [*4].

Table 3 shows RMSE measures for the proposed method

[*4] http://www.vlfeat.org

Table 3 RMSE for two types of SIFT vectors.

	Prop r=1	Conv	Prop r=8
HesAff-SIFT	0.2774	0.2715	0.2594
Grid-SIFT	0.2870	0.2852	0.2740



(a) HesAff-SIFT (b) Grid-SIFT

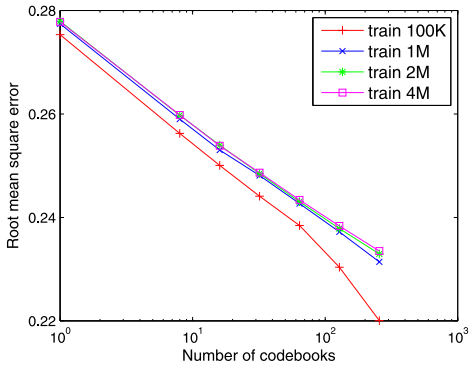
Fig. 6 Assignment matrix obtained by the proposed method where $r = 8$ for HesAff-SIFT and Grid-SIFT features. The (l, i)-th entry of the matrix represents the ratio of $\mathcal{R}_{j,l}$ that satisfies $T_{j,l} = i$ for $1 \le j \le k'$.

where $r = 1, 8$ and the conventional method. In the case of HesAff-SIFT, using different PQ codebooks according to the positions of subvectors (`Conv`) reduces quantization error compared with the case where all subvectors share a single PQ codebook (`Prop` $r = 1$). However, the proposed method where $r = 8$ achieves further improvement over the conventional method by allowing subvectors from different positions to share the same PQ codebooks. In the case of Grid-SIFT, `Conv` achieves almost the same quantization error as `Prop` $r = 1$, while `Prop` $r = 8$ reduces quantization error. This is because the position of the subvectors does not affect the distribution of the subvectors, while the assigned centroid identifiers in coarse quantization do affect the distribution of the subvectors. Note that `Conv` and `Prop` $r = 8$ require the same amount of memory for PQ codebooks. **Figure 6** shows the ratio of residual vector sets at each position that are assigned to each PQ codebook: the (l, i)-th entry of the matrix represents the ratio of $\mathcal{R}_{j,l}$ $(1 \le j \le k')$ that satisfies $T_{j,l} = i$ to k'. In the case of the conventional method, the matrix is identical to the identity matrix. It can be said that the proposed codebook construction algorithm automatically adjusts PQ codebooks and the assignment table to incoming vectors with different statistical properties.

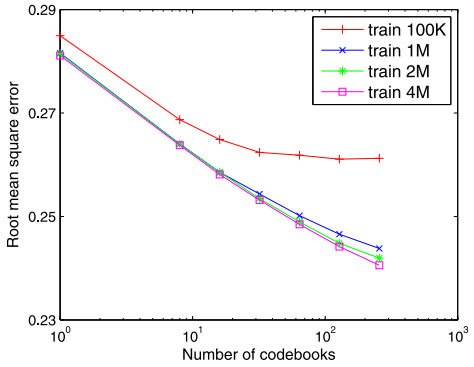
4.4 The Number of PQ Codebooks and Quantization Error

In this section, we evaluate the proposed method in terms of the tradeoff between the number r of PQ codebooks and RMSE. **Figure 7** (a) shows RMSE in the construction of PQ codebooks after convergence. Figure 7 (b) shows RMSE calculated using the reference vectors, which are extracted from a different dataset from the training dataset. This shows that quantization error is reduced in proportion to the logarithm of r. It is also shown that if a sufficient number of training vectors is available, the resulting PQ codebooks are well generalized against non-training vectors. When the number of training vectors is insufficient, increasing the number of PQ codebooks does not contribute to a reduction in the quantization error for non-training vectors.

We also evaluate the relative root mean squared error (RRMSE) for each r to investigate the impact of the reduction of RMSE. We use a random 100K pairs $(\mathbf{x}, \mathbf{y})$ of the reference vectors such that both vectors are assigned to the same centroid in coarse quantization. RRMSE is calculated as



(a) RMSE for the training vectors.



(b) RMSE for the reference vectors.

Fig. 7 RMSE for the training and reference vectors with a different number of training vectors.

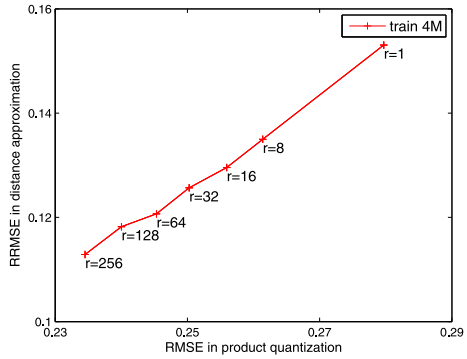


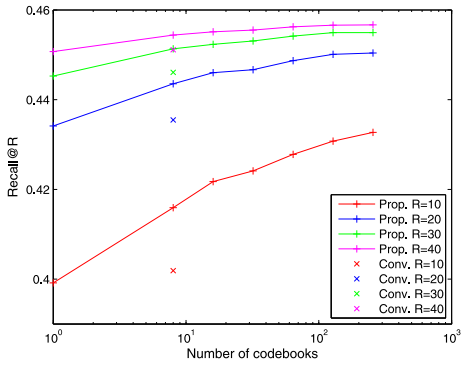
Fig. 8 Relationship between RMSE in product quantization and RRMSE in distance approximation. PQ codebooks created with 4M training vectors are used. RRMSE is calculated for 100K pairs of the reference vectors, which are chosen so that the two reference vectors are assigned to the same centroid in coarse quantization.

$$\mathrm{RRMSE} = \sqrt{E\left[\left(\frac{d(\mathbf{x},\mathbf{y}) - \dot{d}(\mathbf{x},\mathbf{y})}{d(\mathbf{x},\mathbf{y})}\right)^2\right]}. \tag{22}$$

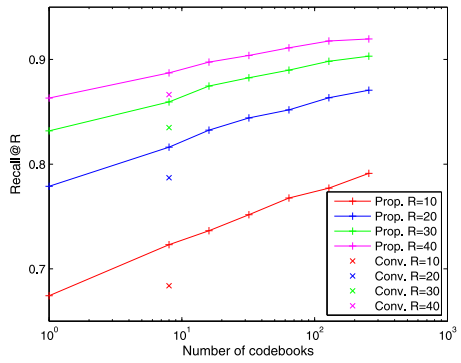
Note that $\dot{d}(\mathbf{x},\mathbf{y})$ is defined by the distance between $\mathbf{x}$ and $q(\mathbf{y})$, which is the quantized version of $\mathbf{y}$. **Figure 8** shows the relationship between the RMSE in product quantization and the RRMSE in distance approximation. Because RRMSE is linearly reduced as RMSE is reduced, it can be said that the reduction of RMSE in product quantization directly contributes to the reduction of actual error in distance appoximation.

4.5 Accuracy of Nearest Neighbor Search

In this section, the approximate NNS accuracy of the proposed method and the conventional method is evaluated using the SIFT



(a) $w = 1$



(b) $w = 16$

Fig. 9 Recall@R for SIFT features as a function of the number r of PQ codebooks obtained by the proposed method and the conventional method.

and GIST features in order to show the effectiveness of the proposed method against different types of vectors. The search quality is measured with Recall@R in the same way as Ref. [13]. That indicates the proportion of query vectors for which the correct nearest neighbor is found in the top-R search results. For each query vector, its nearest neighbor vector among the reference vectors obtained with exact distance calculations is used as a ground truth.

Multiple assignment [13], [20] is also adopted here. Denoting the number of assignments by w, w lists corresponding to the w nearest neighbor centroids in the coarse quantization are searched in an inverted index, instead of searching a single list corresponding to the nearest neighbor centroid. In this case, computational cost in the search procedure becomes w times larger, providing more accurate search results.

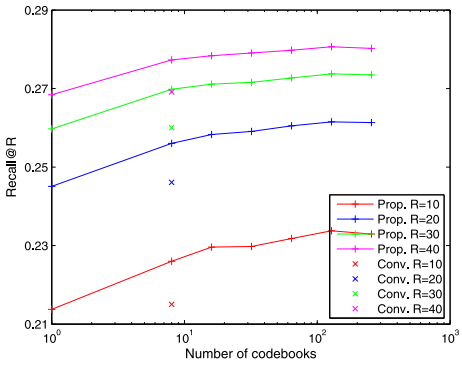
Figure 9 shows Recall@R for 1M SIFT vectors obtained by using a different number of PQ codebooks. It shows that the proposed method improves Recall@R as the logarithm of the number r of PQ codebooks irrespective of R. Comparing the proposed method where $r = 64$ and the conventional method, in the case where $w = 16$, Recall@10 is improved from 0.684 to 0.768, which corresponds to a 12% improvement over the conventional method.

The memory requirement is increased from 33 KB to 262 KB, which corresponds to 2% of the size of the database in this case [*5]. Because the memory requirement for PQ codebooks is indepen-

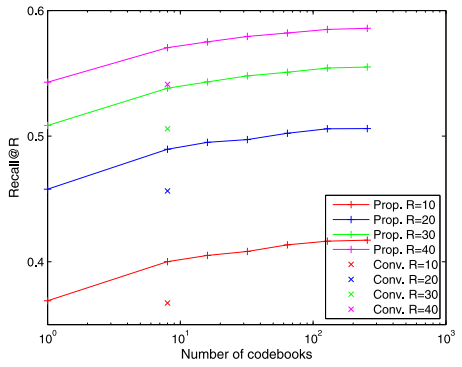
[*5] For each reference feature vector, it requires 4 byte and 8 byte to store an identifier and short code respectively. Therefore, the size of the database (inverted index) is roughly $(4 + 8) \times 1{,}000{,}000$ byte (= 12 MB).

Table 4 Processing times [sec] for coarse quantization (CQ), lookup table construction (LUT), and distance calculation (Dist).

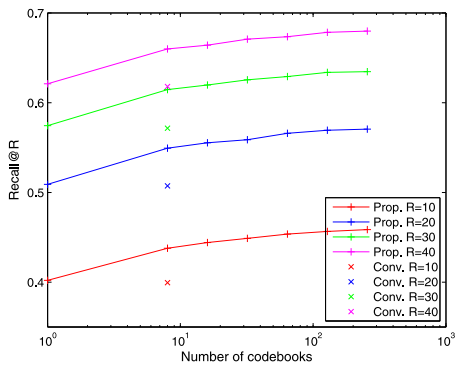
	CQ	LUT	Dist	Total
$w = 1$	3.887	0.773	0.757	5.417
$w = 16$	3.887	11.436	12.160	27.483



(a) $w = 1$



(b) $w = 8$



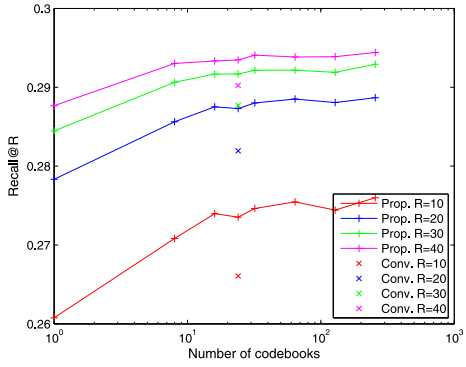
(c) $w = 64$

Fig. 10 Recall@R for GIST features as a function of the number r of PQ codebooks obtained by the proposed method and the conventional method. The number of subvectors m is set to 8.

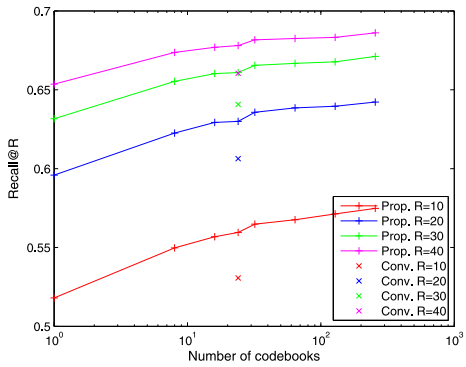
dent of the size of the database, the increase in the amount of memory required becomes negligible as the size of a dataset increases.

Table 4 shows processing times [sec] for coarse quantization (CQ), lookup table construction (LUT), and distance calculation (Dist). They are measured using 100K queries against 1M SIFT vectors with $w = 1$ and $w = 16$. The most time consuming process is coarse quantization in the case where $w = 1$, while it is lookup table construction and distance calculation where $w = 16$ because a computational cost proportional to w is required in lookup table construction and distance calculation.

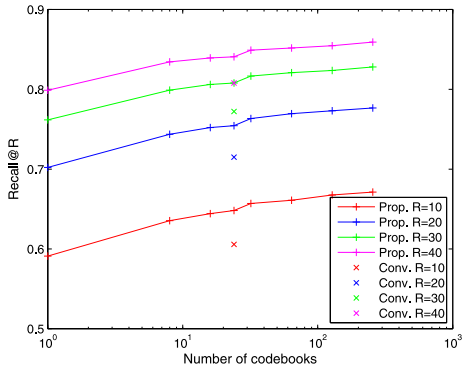
Figure 10 shows Recall@R for 1M GIST vectors obtained using a different number of PQ codebooks, where the number of subvectors is set to 8. We can see a similar tendency to the case where SIFT vectors are used. Comparing the proposed method where $r = 64$ and the conventional method, in the case where $w = 64$, Recall@10 is improved from 0.399 to 0.454, which corresponds to a 13% improvement over the conventional method. Compared with the results for the SIFT dataset, the accuracy is relatively low. This is due to the high-dimensionality of the GIST descriptor; while the dimension of the GIST subvector is three times larger than the SIFT subvector, they are coded with the same length (bits) in the experiments.



(a) $w = 1$



(b) $w = 8$



(c) $w = 64$

Fig. 11 Recall@R for GIST features as a function of the number r of PQ codebooks obtained by the proposed method and the conventional method. The number of subvectors m is set to 24.

Figure 11 shows Recall@R for 1M GIST vectors obtained using a different number of PQ codebooks, where the number of subvectors is increased from 8 to 24. Accordingly, the size of short codes for the GIST vectors is increased from 8 bytes to 24 bytes. This is the same condition as the 8-byte short codes for the SIFT vectors in terms of bits per dimension. From the

Table 5 The average number of searched candidates and Recall@∞ for the SIFT and GIST vectors with different w.

	SIFT		GIST		
	$w = 1$	$w = 16$	$w = 1$	$w = 8$	$w = 64$
# of candidates	1,041	17,245	1,291	10,486	78,724
Recall@∞	0.4585	0.9602	0.2968	0.7218	0.9596

figure, it is found that the proposed method also improves the accuracy compared with the conventional method. Comparing the proposed method with $m = 8$, $r = 8$ (Fig. 10 (c)) and the proposed method with $m = 24$, $r = 24$ (Fig. 11 (c)), in the case where $w = 64$, Recall@10 is significantly improved from 0.438 to 0.648. Therefore, it can be said that the proposed method, as well as the PQ-based approximate NNS, works well for high-dimensional vectors. An important point to be noted is that the accuracy is bounded by Recall@∞ irrespective of the number of subvectors. **Table 5** summarizes the average number of searched candidates and Recall@∞ for the SIFT and GIST vectors with different w. For the GIST vectors, w should be larger than that of the SIFT vectors to obtain the same accuracy, due to the curse of dimensionality.

5. Conclusion

We have proposed an optimized multiple codebook construction algorithm for approximate nearest neighbor search based on product quantization. The algorithm iteratively optimizes both codebooks for product quantization and an assignment table that indicates the optimal codebook in product quantization. Experimental results showed that the proposed method considerably improves the accuracy of approximate nearest neighbor search at the cost of a small increase of the memory required to store the codebooks.

References

[1] Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints, *IJCV*, Vol.60, No.2, pp.91–110 (2004).
[2] Oliva, A. and Torralba, A.: Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope, *IJCV*, Vol.42, No.3, pp.145–175 (2001).
[3] Friedman, J.H., Bentley, J.L. and Finkel, R.A.: An Algorithm for Finding Best Matches in Logarithmic Expected Time, *TOMS*, Vol.3, No.3, pp.209–226 (1977).
[4] Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R. and Wu, A.Y.: An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions, *JACM*, Vol.45, No.6, pp.891–923 (1998).
[5] Andoni, A.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions, *Proc. FOCS*, pp.459–468 (2006).
[6] Philbin, J., Chum, O., Isard, M., Sivic, J. and Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching, *Proc. CVPR*, pp.1–8 (2007).
[7] Silpa-Anan, C. and Hartley, R.: Optimised KD-trees for Fast Image Descriptor Matching, *Proc. CVPR*, pp.1–8 (2008).
[8] Muja, M. and Lowe, D.G.: Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, *Proc. VISAPP*, pp.331–340 (2009).
[9] Beis, J. and Lowe, D.G.: Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces, *Proc. CVPR*, pp.1000–1006 (1997).
[10] Kulis, B. and Darrell, T.: Learning to Hash with Binary Reconstructive Embeddings, *Proc. NIPS*, pp.1042–1050 (2009).
[11] Nistér, D. and Stewénius, H.: Scalable Recognition with a Vocabulary Tree, *Proc. CVPR*, pp.2161–2168 (2006).
[12] Weiss, Y., Torralba, A. and Fergus, R.: Spectral Hashing, *Proc. NIPS*, pp.1753–1760 (2008).
[13] Jégou, H., Douze, M. and Schmid, C.: Product Quantization for Nearest Neighbor Search, *TPAMI*, Vol.33, No.1, pp.117–128 (2011).
[14] Jia, Y., Wang, J., Zeng, G., Zha, H. and Hua, X.S.: Optimizing kd-trees for scalable visual descriptor indexing, *Proc. CVPR*, pp.3392–3399 (2010).
[15] Brandt, J.: Transform Coding for Fast Approximate Nearest Neighbor Search in High Dimensions, *Proc. CVPR*, pp.1815–1822 (2010).
[16] Arthur, D. and Vassilvitskii, S.: k-means++: The Advantages of Careful Seeding, *Proc. SODA*, pp.1027–1035 (2007).
[17] Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T. and Gool, L.V.: A Comparison of Affine Region Detectors, *IJCV*, Vol.60, No.1–2, pp.43–72 (2005).
[18] Torralba, A., Fergus, R. and Freeman, W.T.: 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition, *TPAMI*, Vol.30, No.11, pp.1958–1970 (2008).
[19] Fei-Fei, L. and Perona, P.: A Bayesian hierarchical model for learning natural scene categories, *Proc. CVPR*, pp.524–531 (2005).
[20] Philbin, J., Chum, O., Isard, M., Sivic, J. and Zisserman, A.: Lost in Quantization: Improving Particular Object Retrieval in Large Scale Image Databases, *Proc. CVPR*, pp.1–8 (2008).

Yusuke Uchida received his Bachelor degree of Integrated Human Studies from Kyoto University, Kyoto, Japan, in 2005. He received the degree of Master of Infomatics from Graduate School of Infomatics, Kyoto University, in 2007. His research interests include large-scale content-based multimedia retrieval, augmented reality, and image processing. He is currently with KDDI R&D Laboratories, Inc.

Koichi Takagi received his B.E. and M.E. degree from the Tokyo Institute of Technology in 1996 and 1998 respectively. He has been working at Kokusai Denshin Denwa since 1998 and now is a manager in Technology Strategy Department at KDDI Corporation. From 1998 to 2012, he was with KDDI R&D Laboratories, Inc. and worked in the area of video and audio processing, in particular coding and watermarking. He is a member of IPSJ.

Shigeyuki Sakazawa received his B.E., M.E., and Ph.D. degrees from Kobe University, Japan, all in electrical engineering, in 1990, 1992, and 2005 respectively. He joined Kokusai Denshin Denwa (KDD) Co. Ltd. in 1992. Since then he has been with its R&D Division, and now he is a senior manager of Media and HTML5 Application Laboratory in KDDI R&D Laboratories Inc., Saitama, Japan. His current research interests include video coding, video communication system, image recognition and CG video generation. He received the best paper award from IEICE in 2003.

(Communicated by *Naoko Nitta*)