ITE Trans. on MTA Vol. 6, No. 1, pp. 2-10 (2018)

Invited Paper

A Survey of Product Quantization

Yusuke Matsui †, Yusuke Uchida ††, Hervé Jégou †††,
Shin'ichi Satoh (member)†

Abstract Product Quantization (PQ) search and its derivatives are popular and successful methods for large-scale approximated nearest neighbor search. In this paper, we review the fundamental algorithm of this class of algorithms and provide executable sample codes. We then provide a comprehensive survey of the recent PQ-based methods.

Key words: Product quantization, approximate nearest neighbor search, survey

1. Introduction

With the explosive growth in multimedia data, efficiently indexing and searching large-scale data is a fundamental operation of media search systems. In most situations involving multimedia data, the brute-force exhaustive search has a prohibitive cost both for runtime and memory space. Therefore we focus on the approximated nearest neighbor search (ANN) setup: Given a query vector, our objective is to efficiently find similar vectors from a collection of database vectors, taking into account resources constraints.

Product Quantization (PQ)[26] and its extensions are popular and successful ANN methods for handling large-scale data. Each database vector is quantized into a short code, which we call a PQ-code or simply a code in this paper. The search is conducted over the PQ-codes efficiently using lookup tables. PQ offers three attractive properties: (1) PQ compresses an input vector into a short code (e.g., 64-bits), that enables it to handle typically one billion data points in memory at once; (2) the approximate distance between a raw vector and a compressed PQ code is computed efficiently (the so-called asymmetric distance computation (ADC)), which is a good estimate of the original Euclidean distance; and (3) the data structure and coding algorithms are simple, which allow hybridization with other indexing structures.

The goal of this paper is to provide a comprehensive survey of recent quantization-based techniques, including generalized representations of PQ encoding[4][51], practical large-scale systems with an inverted index structure[30], and hardware-based accelerations[1][29]. In addition, we introduce an executable example of Python codes for PQ, which highlights the simplicity of the data structure of PQ.

The remainder of the paper is organized as follows. We first formulate the search problem in Sec. 2. In Sec. 3, the fundamental algorithm of PQ is reviewed with example codes. Sec. 4 describes variations overs PQ encoding, which generalizes the initial framework. The inverted indexing systems with PQ are reviewed in Sec. 5. Further related applications are introduced in Sec. 6.

2. Approximated Nearest Neighbor Search Problem

In this section, we first define the nearest neighbor search problem, then formulate the approximated nearest neighbor search. Let us denote a database as N D-dimensional vectors $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$, where each $\mathbf{x}_n \in \mathbb{R}^D$. Given a query vector $\mathbf{y} \in \mathbb{R}^D$, the search is formulated as finding the database item that minimizes the distance measure to the query:

$$n^\star = \underset{n \in \{1,\ldots,N\}}{\arg\min} \ \|\mathbf{y} - \mathbf{x}_n\|_2^2. \tag{1}$$

A distance function can be any distance measure such as L_1 norm, etc. We focus on distances in the paper and in particular the distance induced by the Euclidean norm. Yet similarity search may also require to consider other measurements such as cosine similarity, in which case we look for elements maximizing the measure. Eq. (1) can be solved by a simple linear scan, which takes $\mathcal{O}(DN)$ of computational cost with $4DN$ bytes of the memory space*.

Received August 25, 2017; Revised September 24, 2017; Accepted September 28, 2017

† National Institute of Informatics
(Tokyo, Japan)
†† DeNA Co., Ltd.
(Tokyo, Japan)
††† Inria
(Rennes, France)

* In this paper, we assume that a real value is represented by a 32

In real world applications, approximated nearest neighbors (ANN) are often more attracting to achieve better trade-offs between accuracy and resources. Such ANN methods compute an approximation of the results of Eq. (1), trading some accuracy with faster runtime and lower memory usage. In the previous decades, several ANN methods have been developed, including Locality Sensitive Hashing[13)17)] and tree-based methods[8)11)35)], to mention only a few. These particular methods work well for relatively small- or medium-scale data, i.e., up to a few dozen millions elements in the index. However, if the size of the database becomes larger, these approaches require huge storage costs. This does not make them a practical option for handling a large-scale data such as $N = 10^9$.

Recently, there has been a sustained interest on similarity search method employing short-codes. Such methods first convert an input vector into a memory-efficient short code, The search is performed using the resulting short codes. These short-code-based methods are categorized into binary-based[43)] and PQ-based methods. Binary-based methods convert an input vector to a binary vector. The conversion is designed to make the Hamming distance between the two binary strings is a monotonous function of the original distance measure. These two methods are complementary[14)]. Binary-based methods offer a faster distance computation, whereas PQ-based methods achieve a more accurate result for a given memory constraint. In this paper, we review PQ-based methods. Please see[43)46)] for detailed surveys on binary-based methods

3. Product Quantization: Overview

Product Quantization has been originally proposed in the source coding literature[19)]. Jégou et al. applied the idea to the ANN problem[26)] by demonstrating that (1) PQ can be used to compress high-dimensional feature vectors, (2) the distance between an original vector and a PQ-encoded code can be efficiently approximated, and (3) fast search system can be built by combining PQ-encoding and inverted indexing. In this section, we briefly review the algorithm of PQ.

3.1 Encoding

We first show how to encode a real-valued vector into a short code. We employ the following notation in the remainder of the paper. Given a D-dimensional vector $\mathbf{x} \in \mathbb{R}^D$, let us represent it as a concatenation of M sub-vectors.

$$\begin{aligned}\mathbf{x} &= [\underbrace{x_1, x_2, \dots, x_{D/M}}_{\mathbf{x}^{1\top}}, \dots, \underbrace{x_{D-D/M+1}, \dots, x_D}_{\mathbf{x}^{M\top}}]^\top \\ &= \left[\mathbf{x}^{1\top}, \dots, \mathbf{x}^{M\top}\right]^\top, \qquad (2)\end{aligned}$$

where m^{th} sub-vector is denoted as $\mathbf{x}^m \in \mathbb{R}^{D/M}$, for each $m \in \{1, \dots, M\}$.

Next, let us describe how to encode a vector. The idea is to independently encode each sub-vector to an identifier, and to represent the vector as a concatenation of the identifiers. In a training phase, a sub-codebook for each $m \in \{1, \dots, M\}$ is created: $\mathcal{C}^m = \{\mathbf{c}_k^m\}_{k=1}^K$, where we call each $\mathbf{c}_k^m \in \mathbb{R}^{D/M}$ as a sub-codeword. The number K of sub-codewords for each sub-codebook is a parameter specified by a user. $\mathcal{C}^m$ is trained by running the standard k-means clustering over the m^{th} part of the training vectors.

An input vector $\mathbf{x}$ is encoded as follows. First, a sub-encoder for m^{th} sub-vector is introduced; $i^m : \mathbb{R}^{D/M} \to \{1, \dots, K\}$. The m^{th} sub-encoder $i^m(\cdot)$ is a function that returns the identifier of the nearest sub-codeword from $\mathcal{C}^m$. It is formally defined as follows.

$$i^m(\mathbf{x}^m) = \underset{k \in \{1,\dots,K\}}{\arg\min} \|\mathbf{x}^m - \mathbf{c}_k^m\|_2^2. \qquad (3)$$

This search is performed by linearly comparing a D/M-dimensional sub-vector to K sub-codewords, which has a computational complexity of $\mathcal{O}(DK/M)$. Therefore the overall encoding complexity of a given vector is $\mathcal{O}(DK)$.

Next, an encoder, $\mathbf{i} : \mathbb{R}^D \to \{1, \dots, K\}^M$, is defined as a concatenation of M sub-encoders:

$$\mathbf{i}(\mathbf{x}) = \left[i^1\left(\mathbf{x}^1\right), \dots, i^M\left(\mathbf{x}^M\right)\right]^\top. \qquad (4)$$

The encoder breaks a given input vector into M sub-vectors, then applies sub-encoder for each sub-vectors. We call the resulting concatenation of identifiers a PQ-code because the above encoding step is that of a product quantizer. Given an input vector $\mathbf{x}$ and a codebook $\mathcal{C} = \{\mathcal{C}^m\}_{m=1}^M$, the quantization error $e(\mathbf{x}; \mathcal{C})$ in the encoder affects the accuracy in PQ-based ANN, which is routinely measured by the square Euclidean loss as

$$e(\mathbf{x}; \mathcal{C}) = \sum_{m=1}^{M} \min_{k \in \{1,\dots,K\}} \|\mathbf{x}^m - \mathbf{c}_k^m\|_2^2. \qquad (5)$$

3.2 Decoding (reconstruction)

An advantage of PQ is that the original vector can be approximately reconstructed from a PQ-code. Let

bit `float`.

us denote a PQ-code[*] of a vector $\mathbf{x}$ as $\mathbf{i}(\mathbf{x}) = \mathbf{i}_\mathbf{x} = [i^1, \dots, i^M]^\top \in \{1, \dots, K\}^M$. The original vector $\mathbf{x}$ is approximately reconstructed as $\tilde{\mathbf{x}}$ using the PQ-code $\mathbf{i}_\mathbf{x}$ and a decoder $\mathbf{i}^{-1} : \{1, \dots, K\}^M \to \mathcal{C}^1 \times \dots \times \mathcal{C}^M$ as follows.

$$\tilde{\mathbf{x}} = \mathbf{i}^{-1}(\mathbf{i}_\mathbf{x}) = \mathbf{i}^{-1}\left(\begin{bmatrix} i^1 \\ \vdots \\ i^M \end{bmatrix}\right) = \begin{bmatrix} \mathbf{c}^1_{i^1} \\ \vdots \\ \mathbf{c}^M_{i^M} \end{bmatrix}, \tag{6}$$

where $\tilde{\mathbf{x}} \in \mathcal{C}^1 \times \dots \times \mathcal{C}^M \subset \mathbb{R}^D$ is a reconstructed (decoded) D-dimensional vector. Given a PQ-code, the decoder $\mathbf{i}^{-1}(\cdot)$ fetches sub-codewords from the codebook $\mathcal{C} = \mathcal{C}^1 \times \dots \times \mathcal{C}^M$ by using the PQ-codes as indicators. This is an appealing property of PQ-based methods compared to binary-based methods, for which the reconstruction from the original vectors is non-trivial and less accurate and typically discards the vector norm.

3. 3 Memory consumption

PQ is typically employed with aggressive compression rates to ensure that the produced PQ-codes are compact in memory. Because a PQ-code composes of M integers ranging from 1 to K, the memory cost of a PQ-code is $M \log_2 K$ bits. The parameters M and K therefore control the trade-off between reconstruction quality and memory. To represent each integer using `uchar` (8 bit), K is typically set as 256. Then the PQ-code is represented by $8M$ [bit]. Larger values of M lead to better accuracy but slower performance. The performance of PQ-based (and binary-based) methods are often compared by fixing M, for instance by choosing $M = 8$ to compare 64-bit codes.

Since a D-dimensional feature vector is represented by $32D$ bits, a PQ-code with K=256 compresses the original feature by $32D/8M = 4D/M$. For example, 128-dimensional real-valued features represented by 64 bit PQ codes are 64× more memory efficient.

3. 4 Distance computation

Given a query feature vector and a PQ-code, the squared Euclidean distance between the query and the original vector of the PQ-code can be effectively approximated, in particular by the so-called Asymmetric Distance Computation (ADC).

Let us define a query vector as $\mathbf{y} \in \mathbb{R}^D$, and a PQ-code $\mathbf{i}_\mathbf{x} = [i^1, \dots, i^M]^\top \in \{1, \dots, K\}^M$. To approximate the squared distance between the query and the original vector of the PQ-code ($d(\mathbf{y}, \mathbf{x})^2$), we define the Asymmetric Distance $\tilde{d}(\cdot, \cdot)^2$ as the distance between the query and the reconstructed vector from the PQ-code, i.e.,

$$d(\mathbf{y}, \mathbf{x})^2 \approx \tilde{d}(\mathbf{y}, \mathbf{x})^2 = d(\mathbf{y}, \tilde{\mathbf{x}})^2. \tag{7}$$

This can be computed by explicitly reconstructing $\tilde{\mathbf{x}}$ by Eq. (6). However but such a direct computation has a complexity similar to that of a linear search with the original vectors.

The ADC efficiently computes Eq. (7). The computation consists of (1) constructing a distance table (lookup table) and (2) fetching distances. Given $\mathbf{y}$, for each sub-vector $\mathbf{y}^m \in \mathbb{R}^{D/M}$ ($m \in \{1, \dots, M\}$), we compute the distances between $\mathbf{y}^m$ and the K sub-codewords $\mathbf{c}^m_k \in \mathcal{C}^m$, and then store them in the distance table $A : \{1, \dots, M\} \times \{1, \dots, K\} \to \mathbb{R}$. More explicitly, this distance table is defined as follows.

$$A(m, k) = d\left(\mathbf{y}^m, \mathbf{c}^m_k\right)^2. \tag{8}$$

The table A is simply represented as a matrix (2D array). This computation takes $\mathcal{O}(DK)$ operations, and is performed just once per query.

Next, given a database PQ-code $\mathbf{i}_\mathbf{x} = [i^1, \dots, i^M]^\top$, the Asymmetric Distance $\tilde{d}$ is obtained as follows.

$$\begin{aligned} \tilde{d}(\mathbf{y}, \mathbf{x})^2 &= d(\mathbf{y}, \tilde{\mathbf{x}})^2 \\ &= \sum_{m=1}^{M} d\left(\mathbf{y}^m, \mathbf{c}^m_{i^m}\right)^2 = \sum_{m=1}^{M} A(m, i^m). \end{aligned} \tag{9}$$

Because the squared distance is computed for each dimension independently, we can compute separately the square distance of each mth sub-vector. From Eq. (6), the mth sub-vector of the reconstructed vector equals to $\mathbf{c}^m_{i^m}$. The square distance between $\mathbf{y}^m$ and each sub-codeword is already computed and recorded in A, and therefore is fetched by taking the entry identified by m and i^m. Summing the M fetched values provides the Asymmetric Distance. This computation takes only $\mathcal{O}(M)$ table look-ups.

To evaluate the distances for N database PQ-codes, the total computational cost is $\mathcal{O}(DK + MN)$. Typically, the dominant factor is N, which can be a million or even a billion. The construction cost of the distance table can be negligible in many cases.

In summary, with the ADC technique, users can approximate the distances between a query and database PQ-codes without explicitly reconstructing the vectors.

3. 5 Sample codes

One appealing advantage of PQ is its simplicity. Listing 1 shows Python codes of the whole algorithm of PQ,

[*] Note that $\mathbf{i}_\mathbf{x}$ is a constant (a concatenation of integers), not a function. The subscript $\mathbf{x}$ indicates that the PQ-code $\mathbf{i}_\mathbf{x}$ is obtained by encoding a feature vector $\mathbf{x}$.

```python
import numpy as np
from scipy.cluster.vq import vq, kmeans2
from scipy.spatial.distance import cdist

def train(vec, M):
    Ds = int(vec.shape[1] / M) # Ds = D / M
    # codeword[m][k] = $\mathbf{c}_k^m$
    codeword = np.empty((M, 256, Ds), np.float32)

    for m in range(M):
        vec_sub = vec[:, m * Ds : (m + 1) * Ds]
        codeword[m], label = kmeans2(vec_sub, 256)

    return codeword

def encode(codeword, vec): # vec = $\{\mathbf{x}_n\}_{n=1}^N$
    M, _K, Ds = codeword.shape
    # pqcode[n] = $\mathbf{i}(\mathbf{x}_n)$,  pqcode[n][m] = $i^m(\mathbf{x}_n^m)$
    pqcode = np.empty((vec.shape[0], M), np.uint8)

    for m in range(M): # Eq. (3) and Eq. (4)
        vec_sub = vec[:, m * Ds: (m + 1) * Ds]
        pqcode[:, m], dist = vq(vec_sub, codeword[m])

    return pqcode

def search(codeword, pqcode, query):
    M, _K, Ds = codeword.shape
    # dist_table = $A(m, k)$
    dist_table = np.empty((M, 256), np.float32)

    for m in range(M):
        query_sub = query[m * Ds: (m + 1) * Ds]
        dist_table[m, :] = cdist([query_sub],
            ↪ codeword[m], 'sqeuclidean')[0]

    dist = np.sum(dist_table[range(M), pqcode],
      ↪ axis=1)

    return dist

if __name__ == "__main__":
    # Read vec_train, vec ($\{\mathbf{x}_n\}_{n=1}^N$), and query ($\mathbf{y}$)
    M = 4
    codeword = train(vec_train, M)
    pqcode = encode(codeword, vec)
    dist = search(codeword, pqcode, query)
    print(dist)
```

Listing 1: Python scripts of PQ

including training a codebook, encoding vectors, and searching. These sample codes are not pseudo codes but executable scripts. The whole algorithm can be written in only several lines.

3.6 Search system with inverted indexing

The linear scan with ADC is fast compared to the direct linear search, but still slow for a large number of N. To handle million- or even billion-scale search, a search system with inverted indexing was developed in the original PQ paper[26].

As a pre-processing stage, N database items $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ are grouped into J disjoint buckets*, $\mathcal{X}_1, \ldots, \mathcal{X}_J$. This assignment is conducted in the same manner as a coarse-quantization step explained later. Each bucket has a representative vector, $\boldsymbol{\mu}_j \in \mathbb{R}^D$. For each bucket, the difference between the representative vector and each item $\mathbf{x} \in \mathcal{X}_j$ is computed. This residual, $\mathbf{x} - \boldsymbol{\mu}_j$, is encoded into a PQ-code, and stored as a posting-list for each jth bucket.

In the online search step, the system operates in two stages; coarse-quantization and distance-estimation.

* Note that $\mathcal{X}_j$ satisfies (1) $\mathcal{X}_j \subset \mathcal{X}$ for all j, (2) $\mathcal{X} = \bigcup_{j=1}^J \mathcal{X}_j$, and (3) $\mathcal{X}_{j_1} \cap \mathcal{X}_{j_2} = \emptyset$ for all $j_1, j_2 \in \{1, \ldots, J\}$ such that $j_1 \neq j_2$.

• **Coarse-quantization**: Given a query vector $\mathbf{y} \in \mathbb{R}^D$, the nearest bucket $\mathcal{X}_j$ is selected. The residual $\mathbf{y} - \boldsymbol{\mu}_j$ between the query and the representative vector of the jth bucket is computed.

• **Distance-estimation**: The PQ-codes associated with the selected bucket are retrieved by traversing the posting list. The nearest neighbors are then obtained using ADC between the residual vector $\mathbf{y} - \boldsymbol{\mu}_j$ and these PQ-codes.

In this architecture, the search space is restricted by the coarse-quantization. Only the database items in the selected bucket(s) are considered.

Originally, Inverted file system with the asymmetric distance computation (IVFADC) was proposed in the original PQ paper[26]. In IVFADC, the coarse-quantization step is a simple nearest neighbor search for representative vectors. The distance-estimation step is ADC between the query and PQ-codes encoding the residual vectors of the database vectors. This approach can search in $N = 10^9$ data points in 10–100 ms, typically.

In summary, the original paper proposed (1) the PQ-encoding scheme and (2) the search system with inverted indexing. In the next sections, we review some subsequent improvements of PQ-based methods by focusing these two aspects.

4. Generalization and Improvement of PQ Encoding

This section reviews extensions building upon PQ-encoding to improve accuracy or speed. The relation among the methods is visualized in a blue-shaded area in Fig. 1.

4.1 Pre-rotation (OPQ)

The original PQ simply divides an input vector into sub-vectors uniformly, without taking the data distribution into consideration. This works well for a structured vector such as SIFT, but it is not always the case, e.g., D/M-dimensional sub-vectors do not have any meanings. For such general cases, the original paper proposed to apply a random rotation for all vectors preliminary ($\mathbf{x} \leftarrow R\mathbf{x}$, where $R \in \mathbb{R}^{D \times D}$ is a random rotation matrix) to make dimensions uncorrelated. Then the authors[27] proposed to optimize an orthogonal matrix by a set of reflection with a variance balancing criterion.

Optimized Product Quantization (OPQ)[15] extended this idea**. OPQ computes a rotation (orthogonal) ma-

** Note that the same idea was proposed in Cartesian k-means[37] at the same conference independently.

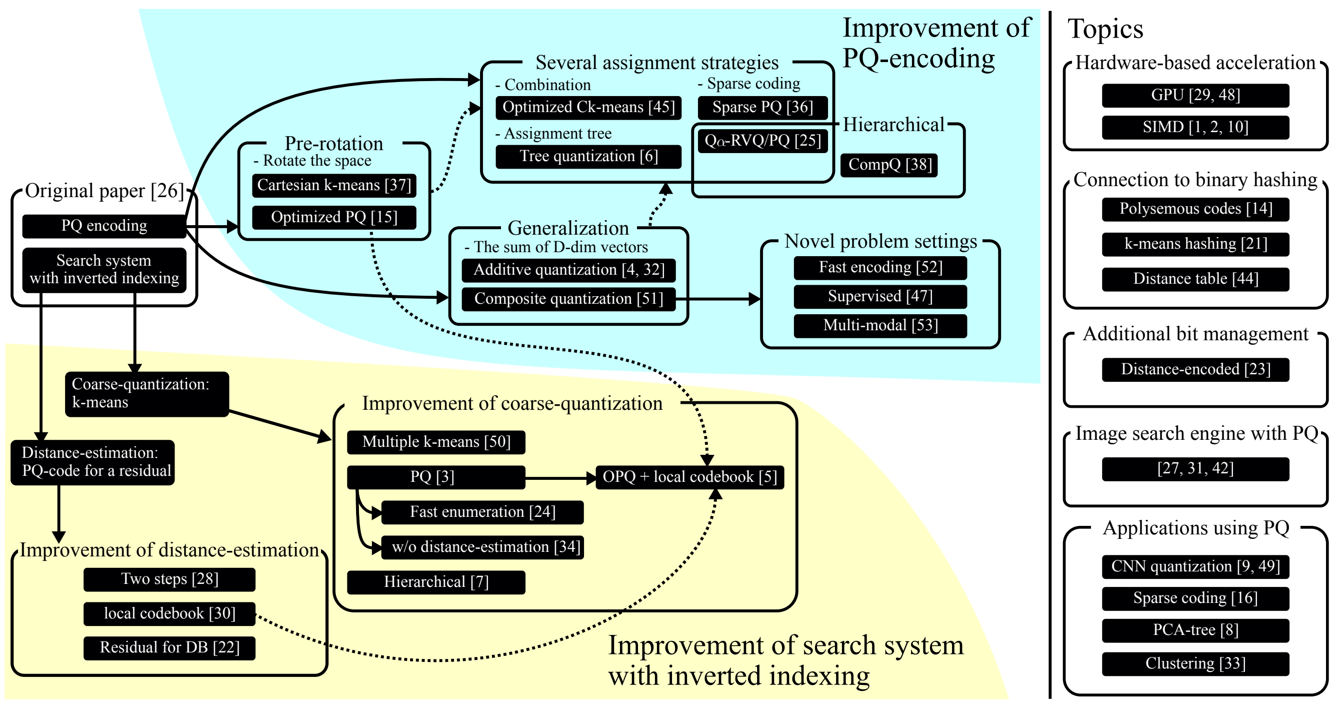

Fig. 1 A relation of PQ-based methods.

trix $R \in \mathbb{R}^{D\times D}$ ($R^\top R = I$) that minimizes the quantization error iteratively. All vectors are rotated by R preliminarily. The training phase of OPQ repeats two steps: (1) All training vectors are rotated by applying R, then codewords are trained as in the same manner with PQ. (2) Given trained codewords, R is updated by solving Orthogonal Procrustes Problem[18]. OPQ is a straightforward extension of PQ, and always boosts the accuracy with a moderate additional cost of applying the rotation matrix. OPQ has been one of the most widely used improvements over PQ.

4.2 Generalization (AQ, CQ)

Additive Quantization (AQ)[4)32)] and Composite Quantization (CQ)[51)] generalize the PQ representation. The original PQ represents a vector as the concatenation of M D/M-dimensional vectors. On the other hand, AQ and CQ represent a vector as the sum of M D-dimensional vectors. The mth sub-codebook is denoted by $\mathcal{C}^m = \{\mathbf{c}^m_k\}_{k=1}^K$ where $\mathbf{c}^m_k \in \mathbb{R}^D$. An input vector $\mathbf{x}$ is still encoded as a vector of indices $\mathbf{i}(\mathbf{x}) = \mathbf{i}_\mathbf{x} = [i^1, \dots, i^M]^\top$, but now the reconstructed vector is obtained as

$$\tilde{\mathbf{x}} = \mathbf{i}^{-1}(\mathbf{i}_\mathbf{x}) = \mathbf{i}^{-1}\left(\begin{bmatrix} i^1 \\ \vdots \\ i^M \end{bmatrix}\right) = \sum_{m=1}^{M} \mathbf{c}^m_{i^m}. \qquad (10)$$

Unlike Eq. (6), the vector is reconstructed by adding the sub-codewords. Note that the encoder $\mathbf{i}(\cdot)$ is no more simple nearest neighbor search for sub-codewords.

If we restrict each sub-codeword such that only values for mth region is nonzero, Eq. (10) becomes Eq. (6). This shows that PQ is a special case of AQ/CQ.

Given a query $\mathbf{y}$ and a AQ/CQ-code $\mathbf{i}_\mathbf{x}$, the asymmetric distance is computed as follows.

$$d(\mathbf{y}, \tilde{\mathbf{x}})^2 = \left\| \mathbf{y} - \sum_{m=1}^{M} \mathbf{c}^m_{i^m} \right\|_2^2 \qquad (11)$$
$$= \|\mathbf{y}\|_2^2 - 2\sum_{m=1}^{M} \mathbf{y}^\top \mathbf{c}^m_{i^m} + \sum_{m_1=1}^{M}\sum_{m_2=1}^{M} (\mathbf{c}^{m_1}_{i^{m_1}})^\top \mathbf{c}^{m_2}_{i^{m_2}}.$$

Here, we can ignore the first term because it is constant for all database items. The second term can be pre-computed and fetched in the similar manner as ADC; creating a lookup tables beforehand ($\mathcal{O}(DKM)$) and fetching values on-line ($\mathcal{O}(M)$). The problem is how to compute the third term. If we take the same strategy for the second term, on-line fetching takes $\mathcal{O}(M^2)$ of the computational cost. This becomes the bottleneck of the computation. AQ and CQ proposed different ways to compute this term efficiently.

In AQ, an additional 1 byte scalar quantization is employed to store the third term. At query time, the value is simply fetched with $\mathcal{O}(1)$. In CQ, the sub-codewords are trained such that the third term (the inner product between sub-codewords) is approximately the same constant value. Note that, in the original PQ, this term is always zero because all sub-codebooks are orthogonal. We can characterize PQ, AQ, and CQ, by the handling of the third term.

Table 1 The comparison of the accuracy for SIFT1M data, 64-bit Codes. The values are from[38].

Method	Recall@1	Recall@10	Recall@100
PQ[26]	0.224	0.599	0.924
CKM/OPQ[15][37]	0.243	0.638	0.940
AQ[4]	0.298	0.741	0.972
CQ[51]	0.288	0.716	0.967
OCK[45]	0.274	0.680	0.945
TQ[6]	0.317	0.748	0.972
CompQ[38]	0.352	0.795	0.987

Because AQ and CQ are generalizations of PQ, the reconstruction error of AQ/CQ is better than that of PQ. On the other hand, training, encoding, and searching of AQ/CQ require more complex steps, which leads more computational cost.

4.3 Alternative coding strategies

To achieve a better accuracy (reconstruction error), several encoding strategies have been proposed. The main idea is to design a better way to assign codewords to an input vector.

Optimized Ck-means[45] extended Cartesian k-means. In PQ, each dimension of a vector is represented by a single codeword. Optimized Ck-means use multiple codewords to represent each dimension to achieve lower reconstruction errors.

Tree quantization (TQ)[6] follows a similar idea. TQ creates an assignment tree that optimizes the best assignment of codewords for each dimension.

Some works borrow the idea of residual encoding[12][38], which is another classical approach for source-coding. CompQ[38] proposed an effective training strategy for residual encoding, and showed that this simple encoding works well for the ANN task. Other extensions aiming at improving the representation power of PQ include sparse-coding for the encoding stage[25][36].

The above strategies usually improve the accuracy of PQ-encoding with the additional assignment cost.

Table 1 shows the comparison of the accuracy among the methods introduced in this section.

4.4 Novel problem settings

Although the original objective of PQ is to solve the general ANN search problem, other settings have been considered in the literature.

How to make the computational cost of per-query pre-processing (not the distance search cost over the database) fast is an open problem, especially for the complex encoding such as AQ/CQ. This is especially important for high-dimensional vectors because per-query pre-processing (building distance tables) depends on D. Sparse Composite Quantization (SCQ)[52] revealed that the second term of Eq. (12) can be efficiently computed if the elements of sub-codewords are sparse. SCQ achieved the same level of the accuracy as CQ with almost the same computational cost of PQ.

Supervised Quantization[47] extends the CQ for a supervised manner. PQ-based methods have been originally developed for an unsupervised setting, whereas the supervised search problem has been handled by binary-based methods. It is reported that Supervised Quantization achieved a better performance compared to binary-based supervised methods, however the setup employed for evaluation has been recently questioned by some researchers[40].

Collaborative Quantization[53] incorporated multi-modal data source such as texts and images. This also outperforms the competitors of binary-based methods.

5. Inverted indexing system with PQ

As described in Sec. 3.6, the original IVFADC consists of two steps, coarse-quantization and distance-estimation. In this section, we review some of the improvements proposed when PQ is combined with an inverted indexing system. These variations are visualized with a yellow-shaded region in Fig. 1.

5.1 Improvement of coarse-quantization

The coarse-quantization step is critical for the runtime performance. IVFADC uses the simple nearest neighbor search for assignment a vector to the coarse centroids, which is the same operation as a standard k-means assignment.

The joint Inverted Indexing[50] proposes a multiple k-means assignment strategy to increase the accuracy. Inverted Multi Indexing (IMI)[3][5] replaces the k-means coarse quantizer by a product quantizer. IMI achieves a good balance between the runtime and the accuracy, and has been widely used. IMI divides the data space as the Cartesian product of two spaces, which amounts to applying PQ on the coarse level with $M = 2$. In the indexing phase, each database item is assigned to one the buckets (Cartesian spaces) using PQ. In the search step, the nearest buckets to a query are enumerated using Multi Sequence Algorithm[3]. The items inside the selected buckets are distance-estimated using ADC.

Several extensions of IMI were proposed. By reducing the search space with upper/lower error bounds, the faster version of Multi Sequence Algorithm was proposed[24]. PQTable[34] presented a method to search without the distance-estimation step.

Another strategy makes use of hierarchical quantizer as a coarse-quantization[7], which especially works well for non-structured data such as deep features (e.g., GoogLeNet activations).

5.2 Improvement of distance-estimation

Next, we review the methods to improve the distance-estimation stage. This step is directly related to the accuracy of the whole system. PQ-encoding for the residual between the representative vector and the data point was used for the original IVFADC.

A two step approach for distance-estimation was proposed[28], where the residual-encoding is computed twice, but at query time only the distances to the most promising candidate vectors are evaluated. A special data structure to refine the error of this encoding step was proposed[22].

Originally, the same sub-codewords were used for encoding the residual associated with the database vectors. It has been shown that learning a product quantizer per inverted list improve the performance[30]. These local sub-codewords for a bucket are created by training only vectors within the bucket. The downside of this approach is that it requires a large set of training data, as well as storing a large volume of centroids.

In the journal version of IMI paper[5], the combination of (1) coarse quantization with IMI, (2) OPQ for both coarse-quantization and distance-estimation, and (3) local codebooks, was proposed. This system is one of the baseline of the state-of-the-art search systems that can handle billion-scale data.

6. Related Topics

In this section, we introduce several topics related to PQ-based methods (the right part in Fig. 1).

First, hardware-based accelerations have been considered. To fully make use of SIMD operations for fast search, methods that fit the data into SIMD registers were proposed[1][2][10] to accelerate the search. With the wide spread of deep learning techniques, GPU is now becoming a common hardware in the computer vision community. Fast search with GPUs begins to be studied[29][48], which might be one of the next promising directions for ANN research.

As reviewed in Sec. 1, binary-based methods are complementary to PQ-based methods. The connection of these two areas was rapidly discussed. Polysemous codes[14] offer the properties of both binary-based and PQ-based methods. The codes are essentially PQ-codes, but the Hamming distance of the two codes reflects the original Euclidean distance as in the binary-based methods. K-means Hashing[21] is one of binary-based methods. We can interpret that K-means Hashing trains a binary code that approximates the PQ-codes, and it is one of the most effective binarization strategy. From the viewpoint of distance-tables, the tables of PQ-based methods are filled by the distances among codewords, whereas those of binary-based methods are filled by the Hamming distances. These relations were intensively investigated[44].

Further additional bits can be added to PQ-code to improve the accuracy[23].

Image search with PQ is one of the main applications using PQ. Several works have evaluated the search performance[27][31][42].

The main idea of PQ is computing distances table among codewords beforehand, such that on-line processing can be efficiently computed by looking up the tables. With PQ-encoding, the data representation becomes memory efficient because the data are presented as a quantized PQ-code. This idea has been widely used in several areas. One of the most active areas is quantization of Deep Neural Networks (DNN). To achieve memory-efficient and fast DNN architecture, several strategies have been intensively researched including pruning[20], factorization[41], binarization[39], sparsifying[41], etc. Quantization is also one of such strategies[9][49]. Another application includes efficient sparse-coding using PQ-based approach[16], fast computation of PCA-tree[8], and clustering[33].

7. Conclusion

In this paper, we introduce the fundamental algorithm of Product Quantization with executable sample codes. We have reviewed several advances on top of the initial technique. We hope this review paper helps to solve a large-scale search problem.

Appendix

This work was supported by JST ACT-I Grant Number JPMJPR16UO and JSPS KAKENHI Grant Number 16H07411.

References

1) F. André, A.-M. Kermarrec, and N. L. Scouarnec. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *Proc. VLDB*, 2015.
2) F. André, A.-M. Kermarrec, and N. L. Scouarnec. Accelerated nearest neighbor search with quick adc. In *Proc. ICMR*, 2017.
3) A. Babenko and V. Lempitsky. The inverted multi-index. In *Proc. IEEE CVPR*, 2012.
4) A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *Proc. IEEE CVPR*, 2014.
5) A. Babenko and V. Lempitsky. The inverted multi-index. *IEEE*

TPAMI, 37(6):1247–1260, 2015.

6) A. Babenko and V. Lempitsky. Tree quantization for large-scale similarity search and classification. In *Proc. IEEE CVPR*, 2015.

7) A. Babenko and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *Proc. IEEE CVPR*, 2016.

8) A. Babenko and V. Lempitsky. Product split trees. In *Proc. IEEE CVPR*, 2017.

9) H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. In *Proc. IEEE CVPR*, 2017.

10) D. W. Blalock and J. V. Guttag. Bolt: Accelerated data mining with fast vector compression. In *Proc. ACM KDD*, 2017.

11) L. Boytsov and B. Naidan. Engineering efficient and effective non-metric space library. In *Proc. SISAP*, 2013.

12) Y. Chen, T. Guan, and C. Wang. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.

13) M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. SCG*, 2004.

14) M. Douze, H. Jégou, and F. Perronnin. Polysemous codes. In *Proc. ECCV*, 2016.

15) T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *IEEE TPAMI*, 36(4):744–755, 2014.

16) T. Ge, K. He, and J. Sun. Product sparse coding. In *Proc. IEEE CVPR*, 2014.

17) A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. VLDB*, 1999.

18) Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE TPAMI*, 35(12):2916–2929, 2013.

19) R. M. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984.

20) S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proc. ICLR*, 2016.

21) K. He, F. Wen, and J. Sun. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In *Proc. IEEE CVPR*, 2013.

22) J.-P. Heo, Z. Lin, X. Shen, J. Brandt, and S.-E. Yoon. Shortlist selection with residual-aware distance estimator for k-nearest neighbor search. In *Proc. CVPR*, 2016.

23) J.-P. Heo, Z. Lin, and S.-E. Yoon. Distance encoded product quantization. In *Proc. IEEE CVPR*, 2014.

24) M. Iwamura, T. Sato, and K. Kise. What is the most efficient way to select nearest neighbor candidates for fast approximate nearest neighbor search? In *Proc. IEEE ICCV*, 2013.

25) H. Jain, P. Pérez, R. Gribonval, J. Zepeda, and H. Jégou. Approximate search with quantized sparse representations. In *Proc. ECCV*, 2016.

26) H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011.

27) H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Proc. IEEE CVPR*, 2010.

28) H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *Proc. IEEE ICASSP*, 2011.

29) J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017.

30) Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *Proc. IEEE CVPR*, 2014.

31) J. Li, X. Lan, X. Li, J. Wang, N. Zheng, and Y. Wu. Online variable coding length product quantization for fast nearest neighbor search in mobile retrieval. *IEEE TMM*, 19(3):559–570, 2017.

32) J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In *Proc. ECCV*, 2016.

33) Y. Matsui, K. Ogaki, T. Yamasaki, and K. Aizawa. Pqk-means: Billion-scale clustering for product-quantized codes. In *Proc. MM*, 2017.

34) Y. Matsui, T. Yamasaki, and K. Aizawa. Pqtable: Fast exact asymmetric distance neighbor search for product quantization using hash tables. In *Proc. IEEE ICCV*, 2015.

35) M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE TPAMI*, 36(11):2227–2240, 2014.

36) Q. Ning, J. Zhu, Z. Zhong, S. C. H. Hoi, and C. Chen. Scalable image retrieval by sparse product quantization. *IEEE TMM*, 19(3):586–597, 2017.

37) M. Norouzi and D. J. Fleet. Cartesian k-means. In *Proc. IEEE CVPR*, 2013.

38) E. C. Ozan, S. Kiranyaz, and M. Gabbouj. Competitive quantization for approximate nearest neighbor search. *IEEE TKDE*, 28(11):2884–2894, 2016.

39) M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. ECCV*, 2016.

40) A. Sablayrolles, M. Douze, N. Usunier, and H. Jégou. How should we evaluate supervised hashing? In *Proc. IEEE ICASSP*, pages 1732–1736. IEEE, 2017.

41) T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proc. IEEE ICASSP*, 2013.

42) E. Spyromitros-Xioufis, S. Papadopoulos, I. Y. Kompatsiaris, G. Tsoumakas, and I. Vlahavas. A comprehensive study over vlad and product quantization in large-scale image retrieval. *IEEE TMM*, 16(6):1713–1728, 2014.

43) J. Wang, W. Liu, S. Kumar, and S.-F. Chang. Learning to hash for indexing big data - a survey. *Proc. IEEE*, 2015.

44) J. Wang, H. T. Shen, S. Yan, N. Yu, S. Li, and J. Wang. Optimized distances for binary code ranking. In *Proc. MM*, 2014.

45) J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li. Optimized cartesian k-means. *IEEE TKDE*, 27(1):180–192, 2015.

46) J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *CoRR*, abs/1606.00185, 2016.

47) X. Wang, T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Supervised quantization for similarity search. In *Proc. CVPR*, 2016.

48) P. Wieschollek, O. Wang, A. Sorkine-Hornung, and H. P. A. Lensch. Efficient large-scale approximate nearest neighbor search on the gpu. In *Proc. CVPR*, 2016.

49) J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proc. IEEE CVPR*, 2016.

50) Y. Xia, K. He, F. Wen, and J. Sun. Joint inverted indexing. In *Proc. IEEE ICCV*, 2013.

51) T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *Proc. ICML*, 2014.

52) T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Sparse composite quantization. In *Proc. IEEE CVPR*, 2015.

53) T. Zhang and J. Wang. Collaborative quantization for cross-modal similarity search. In *Proc. CVPR*, 2016.

Yusuke Matsui received the B.E, M.A., and Ph.D. in information science and technology from the University of Tokyo in 2011, 2013, and 2016, respectively. He is currently a postdoctoral researcher at National Institute of Informatics, Japan. His research interests lie in computer vision, computer graphics, and multimedia processing, with particular interest in image retrieval.

Yusuke Uchida received his B.E. and M.E. from Kyoto University, Kyoto, Japan, in 2005 and 2007 respectively. He received his Ph.D. degree in information science and technology from the University of Tokyo in 2016. His research interests include large-scale content-based multimedia retrieval and deep learning-based image recognition. He is currently with DeNA Co., Ltd.

Hervé Jégou is a Research Scientist and Manager at Facebook AI Research since 2015. He is a former student of the Ecole Normale Supérieure de Cachan, holding a M.S. (2002) and PhD (2005) from University of Rennes I. During his PhD, he worked on source/channel coding. After that, he turned out to Computer Vision and Pattern Recognition and joined INRIA as a permanent researcher in 2006, mostly working and leading several projects related to large image and video collections. His current interests at FAIR include large-scale processing, computer vision and natural language processing.

Shin'ichi Satoh is a professor at National Institute of Informatics (NII), Tokyo. He received PhD degree in 1992 at the University of Tokyo. His research interests include image processing, video content analysis and multimedia database. Currently he is leading the video processing project at NII, addressing video analysis, indexing, retrieval, and mining for broadcasted video archives.