Asymmetric Mapping Quantization for Nearest Neighbor Search

Weixiang Hong, Xueyan Tang, Jingjing Meng and Junsong Yuan

Abstract—Nearest neighbor search is a fundamental problem in computer vision and machine learning. The straightforward solution, linear scan, is both computationally and memory intensive in large scale high-dimensional cases, hence is not preferable in practice. Therefore, there have been a lot of interests in algorithms that perform approximate nearest neighbor (ANN) search. In this paper, we propose a novel addition-based vector quantization algorithm, Asymmetric Mapping Quantization (AMQ), to efficiently conduct ANN search. Unlike existing addition-based quantization methods that suffer from handling the problem caused by the norm of database vector, we map the query vector and database vector using different mapping functions to transform the computation of L-2 distance to inner product similarity, thus do not need to evaluate the norm of database vector. Moreover, we further propose Distributed Asymmetric Mapping Quantization (DAMQ) to enable AMQ to work on very large dataset by distributed learning. Extensive experiments on approximate nearest neighbor search and image retrieval validate the merits of the proposed AMQ and DAMQ.

Index Terms—Vector Quantization, Nearest Neighbour Search, Image Retrieval, Distributed Optimization.

1 Introduction

Nearest neighbor (NN) search has wide applications in computer vision, machine learning and information retrieval, *e.g.*, image retrieval, object instance search and k-NN classifier [31], *etc.* The straightforward solution, linear scan, is both computationally and memory intensive in large scale high-dimensional cases, hence is not preferable in practice. Consequently, there have been a lot of interests in algorithms that perform approximate nearest neighbor (ANN) search.

ANN search is traditionally addressed with hashing methods that have been comprehensively surveyed in [33]. However, a family of methods based on vector quantization [2], [11], [20], [40], [41] has recently triggered interests from computer vision, machine learning and multimedia retrieval communities, due to its superior accuracy and comparable efficiency compared with hashing techniques. Different from hashing, these methods perform ANN search by learning numerous decimal quantizers, hence do not suffer from the quantization loss from decimal space to binary space, which is the main reason of their better accuracy compared with hashing. Meanwhile, quantization-based methods are also efficient thanks to the smart use of lookup tables.

Vector quantization algorithms can be coarsely classified into two categories: (1) partition-based quantization and (2) addition-based quantization. Product quantization (PQ) is perhaps the most well-known work that belongs to partition-based quantization [8], [11], [20], [27], [37]. PQ partitions the feature space into a number of disjoint subspaces and quantizes each subspace separately. A vector is represented by a short code composed of its subspace quantization indices. The Euclidean distance between two vectors can be efficiently estimated from their codes using a lookup table. There have been several attempts to improve product quantization, such as distance-encoded product quantization [11], optimized product quantization [8] and Cartesian k-means (CKM) [27]. On the other hand, additive quantization (AQ) is one of the pioneer works for addition-based quantization [1], [2], [40], [41]. AQ improves the vector quantization accuracy by approximating a database vector using the addition of dictionary words selected from different dictionaries. Since the encoding phase in AQ is essentially high-order Markov Random Fileds (MRFs) and NP-hard, several recent works [2], [4], [25] have been proposed to address this drawback. Composite Quantization (CQ) [40] is another kind of addition-based quantization. CQ approximates a database vector using the summation of dictionary words with the constant inter-dictionary-element-product constraint.

Generally speaking, addition-based quantization methods perform better than partition-based ones, because addition-based quantizations do not decompose data space into orthogonal subspaces and thus make no subspace independence assumptions. Nevertheless, an inherent flaw to addition-based approaches is that the norm of database vector cannot be efficiently estimated. Specifically, the L-2 distance between the query q and any database vector x can be expanded as:

$$||\mathrm{q}-\mathrm{x}||_2^2 = ||\mathrm{q}||_2^2 - 2\mathrm{q}^T\mathrm{x} + ||\mathrm{x}||_2^2. \quad (1)$$

The first term is identical to all database vectors and can be omitted. Supposing x is approximated by addition-based quantization approximation such as $\mathrm{x} \approx \sum_{m=1}^{M} \mathrm{C}_{mk_m}$, where C_{mk_m} represents the k_m-th element from dictionary C_m, then the second item can be approximately computed by lookup table. Unfortunately, the third item $||\mathrm{x}||_2^2$ cannot be fast computed with addition-based codes. Existing methods tackle this problem following two different kinds of strategies. The first one adopted by AQ [1], LSQ [25] and ArborC [4] utilizes additional memory to encode $||\mathrm{x}||_2^2$,

- *Weixiang Hong is with National University of Singapore. Xueyan Tang is with Nanyang Technological University. Jingjing Meng and Junsong Yuan are with State University of New York at Buffalo.*
- *Correspondence to Junsong Yuan at jsyuan@buffalo.edu.*

which is straightforward but not scalable since the memory request grows linearly to the number of database vectors. The second one adopted by CQ [40] and SQ [41] reformulates Equation (1) and enforces an inter-dictionary-element-product constraint on the dictionary C to reduce the computation (See Section 2 for details). Unluckily, the extra constraint will limit the model capacity and hurt the performance [40].

In this work, we propose a novel solution, Asymmetric Mapping Quantization (AMQ), to elegantly eliminate the computation of database vector norm. By appending an additional dimension to both database vectors and query vectors, AMQ transforms the computation of L-2 distance to inner product similarity, which does not involve the computation of $||\mathrm{x}||_2^2$. Meanwhile, the inner product similarity can be efficiently computed using table lookups in the context of addition-based quantization. We note that similar approach has been proposed in [32]. Since we do not enforce any constraint on the dictionary C, the objective function in our optimization problem is convex on C and has a closed-form solution. As shown in Section 4, the convexity on C further permits us to learn AMQ under a distributed setting by the alternating direction method of multipliers (ADMM) [12]. We conduct extensive experiments on approximate nearest neighbor search and image retrieval, and the empirical results validate the merits of the proposed approach.

2 Backgrounds

In this section, we briefly introduce Composite Quantization [34], [40], [41] and Additive Quantization [1], [4], [25], which are essential to the understanding of our work.

2.1 Composite Quantization

Composite Quantization [40] aims to approximate a vector $\mathrm{x} \in \mathbb{R}^d$ by the composition of M vectors $\{\mathrm{C}_{1k_1}, \mathrm{C}_{2k_2}, \ldots, \mathrm{C}_{Mk_M}\}$, each of which is selected from a dictionary with K elements, *i.e.*, C_{mk_m} is the k_mth element in the dictionary C_m, and $\forall m \in \{1, 2, \ldots, M\}$, $\mathrm{C}_m = \{\mathrm{C}_{m1}, \mathrm{C}_{m2}, \ldots, \mathrm{C}_{mK}\}$.

Let $\bar{\mathrm{x}} = \sum_{m=1}^{M} \mathrm{C}_{mk_m}$ be the approximation of vector x. The accuracy of nearest neighbor search relies on the quality of the distance approximation, *i.e.*, how small is the difference between the distance of the query q to the vector x and the distance to the approximation $\bar{\mathrm{x}}$. According to the triangle inequality, $\big|||\mathrm{q}-\mathrm{x}||_2 - ||\mathrm{q}-\bar{\mathrm{x}}||_2\big| \leq ||\mathrm{x}-\bar{\mathrm{x}}||_2$, the distance approximation error in ANN search is bounded by the vector approximation error, which is formulated as follows,

$$\min_{\mathrm{C}_{mk_m}} \sum_{\mathrm{x}\in\mathrm{X}} ||\mathrm{x} - \sum_{m=1}^{M} \mathrm{C}_{mk_m}||_2^2, \tag{2}$$

where C_{mk_m} is the selected element from the dictionary C_m for the database vector x.

With the approximation $\bar{\mathrm{x}} = \sum_{m=1}^{M} \mathrm{C}_{mk_m}$, we have

$$\begin{aligned} ||\mathrm{q}-\bar{\mathrm{x}}||_2^2 = &\sum_{m=1}^{M} ||\mathrm{q} - \mathrm{C}_{mk_m}||_2^2 \\ &- (M-1)||q||_2^2 + \sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M} \mathrm{C}_{ik_i}^T \mathrm{C}_{jk_j}. \end{aligned} \tag{3}$$

Given the query q, the first term can be efficiently computed using lookup tables, and the second term is constant for all database vectors, hence unnecessary to compute for ANN search. For the third term, we constrain it to be a constant ϵ, *i.e.*, $\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M} \mathrm{C}_{ik_i}^T \mathrm{C}_{jk_j} = \epsilon$, which is referred to as constant inter-dictionary-element-product in Composite Quantization.

Let $\mathrm{C} = [\mathrm{C}_1\mathrm{C}_2\cdots\mathrm{C}_M] \in \mathbb{R}^{d\times MK}$ be the whole dictionary, $\mathrm{B} = [\mathrm{b}_1^T\mathrm{b}_2^T\cdots\mathrm{b}_N^T] \in \mathbb{R}^{MK\times N}$ be the matrix consisting of all codes. Furthermore, we impose three extra constraints on B to fully complete the formulation, namely, $\mathrm{b}_n = [\mathrm{b}_{n1}\mathrm{b}_{n2}\cdots\mathrm{b}_{nM}] \in \{0,1\}^{MK}$, $\mathrm{b}_{nm} \in \{0,1\}^K$ and $||\mathrm{b}_{nm}||_1 = 1$. Finally, the optimization problem is formulated as:

$$\begin{aligned} \min_{\mathrm{C},\mathrm{B},\epsilon} \quad & ||\mathrm{X} - \mathrm{CB}||_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \mathrm{B} = [\mathrm{b}_1^T\mathrm{b}_2^T\cdots\mathrm{b}_N^T], \mathrm{b}_n = [\mathrm{b}_{ni}^T\mathrm{b}_{ni}^T\cdots\mathrm{b}_{ni}^T]^T \\ & \mathrm{b}_{nm} \in \{0,1\}^K, ||\mathrm{b}_{nm}||_1 = 1 \\ & \sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M} \mathrm{b}_{ni}^T\mathrm{C}_i^T\mathrm{C}_j\mathrm{b}_{nj} = \epsilon \\ & n = 1, 2, \ldots, N, m = 1, 2, \ldots, M, \end{aligned} \tag{4}$$

which is hard to directly solve. CQ relaxes the constant inter-dictionary-element-product constraint as:

$$\begin{aligned} \min_{\mathrm{C},\mathrm{B},\epsilon} \quad & ||\mathrm{X} - \mathrm{CB}||_{\mathrm{F}}^2 + \mu(\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M} \mathrm{b}_{ni}^T\mathrm{C}_i^T\mathrm{C}_j\mathrm{b}_{nj} - \epsilon)^2 \\ \text{s.t.} \quad & \mathrm{B} = [\mathrm{b}_1^T\mathrm{b}_2^T\cdots\mathrm{b}_N^T], \mathrm{b}_n = [\mathrm{b}_{ni}^T\mathrm{b}_{ni}^T\cdots\mathrm{b}_{ni}^T]^T \\ & \mathrm{b}_{nm} \in \{0,1\}^K, ||\mathrm{b}_{nm}||_1 = 1 \\ & n = 1, 2, \ldots, N, m = 1, 2, \ldots, M, \end{aligned} \tag{5}$$

and solves the dictionary C in (5) by a quasi-Newton algorithm (L-BFGS [26]). Unfortunately, (5) is not convex on C, thus the L-BFGS algorithm cannot guarantee to reach the global optimum of (5), which may result in undesirable performance loss.

2.2 Additive Quantization

Similar to Composite Quantization, Additive Quantization [1] also expresses each vector x as the summation of several dictionary elements $\mathrm{x} \approx \sum_{m=1}^{M} \mathrm{C}_{mk_m}$, and aims to minimize the vector approximation error, *i.e.*, Equation (2). The key difference between CQ and AQ is the way to estimate the distance between query vector and database vectors. Specifically, AQ expands $||\mathrm{q}-\mathrm{x}||_2^2$ as:

$$||\mathrm{q}-\mathrm{x}||_2^2 = ||\mathrm{q}||_2^2 - 2\mathrm{q}^T\mathrm{x} + ||\mathrm{x}||_2^2. \tag{6}$$

The first term $||\mathrm{q}||_2^2$ is omitted similar to CQ. The second term $\mathrm{q}^T\mathrm{x}$ can be approximately estimated using table lookup, *i.e.*

$$\mathrm{q}^T\mathrm{x} \approx \mathrm{q}^T\bar{\mathrm{x}} = \sum_{m=1}^{M} \mathrm{q}^T\mathrm{C}_{mk_m}. \tag{7}$$

The computational cost of this step for one database vector is $\mathcal{O}(M)$.

For the third item $||\mathrm{x}||_2^2$, AQ provides two different solutions to estimate it. The first one is to non-uniformly quantize the scalar value of $||\mathrm{x}||_2^2$, which results in additional

memory cost that grows linearly to the database size. The other way is to approximately estimate $||\mathrm{x}||_2^2$ using table lookups, *i.e.*,

$$||\mathrm{x}||_2^2 \approx ||\bar{\mathrm{x}}||_2^2 = ||\sum_{m=1}^{M} \mathrm{C}_{mk_m}||_2^2 = \sum_{i=1}^{M}\sum_{j=1}^{M} \mathrm{C}_{ik_i}^T \mathrm{C}_{jk_j}. \quad (8)$$

Although Equation (8) does not cost additional memory footprint, the computational cost of Equation (8) for one database vector is $\mathcal{O}(M^2)$, which grows quadratically with M and can slow down the computation of distance considerably [1].

3 Asymmetric Mapping Quantization

In this section, we introduce the Asymmetric Mapping Quantization, which elegantly addresses the drawback of CQ based methods [34], [40], [41] and AQ based approaches [1], [4], [25].

3.1 Asymmetric Mapping

First, we define two vector transformation functions $P : \mathbb{R}^d \mapsto \mathbb{R}^{d+1}$ and $Q : \mathbb{R}^d \mapsto \mathbb{R}^{d+1}$ as follows:

$$\begin{aligned} P(\mathrm{x}) &= [\mathrm{x}; ||\mathrm{x}||_2^2], \\ Q(\mathrm{q}) &= [\mathrm{q}; -\frac{1}{2}]. \end{aligned} \quad (9)$$

where $[;]$ is the concatenation operation. $P(\mathrm{x})$ appends the 2-norm $||\mathrm{x}||_2^2$ at the end of the vector x, while $Q(\mathrm{q})$ simply extends the vector q by an extra $-\frac{1}{2}$. By observing that

$$\begin{aligned} -2Q(\mathrm{q})^T P(\mathrm{x}) &= -2[\mathrm{q}; -\frac{1}{2}]^T [\mathrm{x}; ||\mathrm{x}||_2^2] \\ &= -2\mathrm{q}^T\mathrm{x} + ||\mathrm{x}||_2^2 \\ &= ||\mathrm{q}-\mathrm{x}||_2^2 - ||\mathrm{q}||_2^2, \end{aligned} \quad (10)$$

Since the 2-norm of the query $||\mathrm{q}||_2^2$ is constant to any database vector, we have:

$$\arg\min_{\mathrm{x}\in X} ||\mathrm{q}-\mathrm{x}||_2^2 = \arg\max_{\mathrm{x}\in X} Q(\mathrm{q})^T P(\mathrm{x}). \quad (11)$$

This gives us the connection between solving maximum inner product search and approximate nearest neighbor search. To be more specific, if one needs to compute the rank list of a dataset X with respect to a query q for L-2 distance, it is feasible to first compute the rank list of $P(\mathrm{X})$ with respect to $Q(\mathrm{q})$ for inner product similarity, followed by reversing the rank list.

3.2 Fast Inference for Inner Product Similarity

Let $\bar{P(\mathrm{x})} = \sum_{m=1}^{M} \mathrm{C}_{mk_m}$ be the approximation of vector $P(\mathrm{x})$, where $\mathrm{C}_{mk_m} \in \mathbb{R}^{d+1}$, $P(\mathrm{x})$ is the mapping from the database vector $x \in \mathbb{R}^d$. With this approximation, we have

$$\begin{aligned} Q(\mathrm{q})^T \bar{P(\mathrm{x})} &= Q(\mathrm{q})^T \sum_{m=1}^{M} \mathrm{C}_{mk_m} \\ &= \sum_{m=1}^{M} Q(\mathrm{q})^T \mathrm{C}_{mk_m}, \end{aligned} \quad (12)$$

which can be efficiently computed using lookup tables. After the rank list for inner product similarity is obtained, one can simply reverse it to get the rank list for L-2 distance. Our inference scheme shares the spirit of asymmetric linear scan [20].

3.3 Objective Function

To boost the ANN search performance under L-2 distance, one needs to improve the quality of inner product similarity search, which is conditioned on the approximation quality of the compositional codes [7]. Therefore, our optimization objective is:

$$\begin{aligned} \min_{\mathrm{C,B}} \quad & ||\mathrm{Y} - \mathrm{CB}||_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \mathrm{B} = [\mathrm{b}_1^T \mathrm{b}_2^T \cdots \mathrm{b}_N^T], \mathrm{b}_n = [\mathrm{b}_{n1}^T \mathrm{b}_{n2}^T \cdots \mathrm{b}_{nM}^T]^T \\ & \mathrm{b}_{nm} \in \{0,1\}^K, ||\mathrm{b}_{nm}||_1 = 1 \\ & n = 1, 2, \ldots, N, m = 1, 2, \ldots, M. \end{aligned} \quad (13)$$

where $\mathrm{Y} = P(\mathrm{X}) \in \mathbb{R}^{(d+1)\times N}$. Each dictionary element should also be one-dimensional longer, *i.e.*, $\mathrm{C} \in \mathbb{R}^{(d+1)\times MK}$. The shape of the compositional matrix remains unchanged as $\mathrm{B} \in \{0,1\}^{MK\times N}$. By comparing Equation (13) and (4), one can immediately observe that the constant inter-dictionary-element-product constraint has been removed because the Euclidean distance between the query and database vectors can now be efficiently computed via Equation (12) without posing this constraint. Consequently, given B fixed, the dictionary C can be optimally solved since Equation (13) is quadratic on C.

3.4 Optimization

The problem formulated in (13) is a mixed-binary-integer program, which consists of two groups of unknown variables: dictionaries C and composition matrix B. We use the alternative optimization technique to iteratively solve the problem. Each iteration alternatively updates B and C. The details are given below.

3.4.1 *Update* C

Because Equation (13) is quadratic on C, a closed-form solution for updating C can be easily derived as:

$$\mathrm{C} = \mathrm{YB}^T(\mathrm{BB}^T)^{-1}. \quad (14)$$

3.4.2 *Update* B

Let $\mathrm{y}_n \in \mathbb{R}^{d+1}$ and $\mathrm{b}_n \in \{0,1\}^{MK}$ denote the n-th vector in Y and B, respectively. Given C fixed, it can be easily found that b_n is independent to $\{\mathrm{b}_l\}_{l\neq n}$ for all the other vectors. Thus, the optimization problem is decomposed to N subproblems,

$$\begin{aligned} \min_{\mathrm{b}_n} \quad & ||\mathrm{y}_n - \mathrm{C}\mathrm{b}_n||^2 \\ \text{s.t.} \quad & \mathrm{b}_n = [\mathrm{b}_{n1}^T \mathrm{b}_{n2}^T \cdots \mathrm{b}_{nM}^T]^T \\ & \mathrm{b}_{nm} \in \{0,1\}^K, ||\mathrm{b}_{nm}||_1 = 1 \\ & m = 1, 2, \ldots, M. \end{aligned} \quad (15)$$

The problem is essentially a high-order MRF problem and NP-hard. To efficiently find a feasible local optimum, we leverage the Stochastic local search (SLS) method suggested by [25]. For the local search procedure of SLS, we iteratively update the M subvectors $\{\mathrm{b}_{nm}\}$ in turn. Specifically, given $\{\mathrm{b}_{nl}\}_{l\neq m}$ fixed, we exhaustively check all the elements in the dictionary C_m, and find the element that minimizes the objective function value in Equation (15), then update $\{\mathrm{b}_{nm}\}$ by setting the corresponding entry in b_{nm} to be 1 and

all the others to be 0. For the perturbation procedure of SLS, we perturb the results of local search procedure by replacing k segments of b_{nm} with the one-hot coded samples from the uniform distribution $\mathcal{U}(1, M)$. Following [25], the perturbed solution is used as the starting point for the subsequent local search procedure if it decreases more objective value in Equation (15) than the original solution.

3.5 Implementation Details

In practical implementations, we use the transformations as follows:

$$\begin{aligned} P(\mathrm{x}) &= [\mathrm{x}; \frac{||\mathrm{x}||_2^2}{d^2}], \\ Q(\mathrm{q}) &= [\mathrm{q}; -\frac{d^2}{2}], \end{aligned} \tag{16}$$

where d is the feature vector dimension. This is to prevent the magnitude of vector norm from dominating the objective value, while the rank of inner product similarity in inference stage will not be affected.

Unlike CQ based methods [40], [41], our AMQ is hyperparameter-free. By comparing Equation (5) and (13), one can observe that our formulation does not require the penalty weight μ. Different from AQ based approaches [1], [4], [25], our AMQ does not require additional memory footprint or computational resource in the inference stage as discussed in Section 3.2.

4 Distributed Asymmetric Mapping Quantization

Despite the good performance of AMQ, it is designed for the centralized setting, or in other words, are single-machine approaches. Nevertheless, due to the explosion in size and complexity of modern datasets, more and more real-world applications need to deal with data distributed across different locations, such as distributed databases [6], images/videos over the networks [9], *etc.* Furthermore, in some applications, the data is inherently distributed. For example, in video surveillance [10] and sensor networks [17], the data is collected at distributed sites. In such contexts, the quantizers should be learned based on the entire dataset in order to get unbiased quantization codes for the data.

One intuitive way is to gather all data together at a central server before training, but it is not a feasible option because of the huge communication overhead. Besides, directly training on large-scale data is often prohibitive in both time and space, which further prevents it from practical applications. As a consequence, it is important to develop quantization algorithms that are both powerful enough to capture the complexity of large scale data, and scalable enough to process huge datasets in parallel. However, to our knowledge, this critical and challenging problem has rarely been explored in the literature. We propose Distributed Asymmetric Mapping Quantization (DAMQ) for data that is distributed across different nodes of an arbitrary network (*e.g.*, Figure 1). Unlike the conventional centralized methods which require gathering the distributed data from all nodes to learn common quantizers, our method learns such quantizers in a distributed manner. Each node learns a set of dictionaries on its local data, and only exchanges

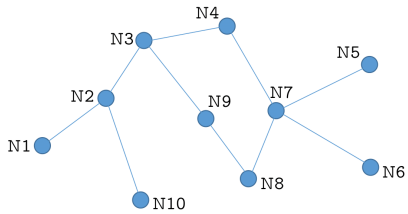

Fig. 1. A randomly generated network with 10 nodes. Such a network can be modeled with an undirected and connected graph.

the local dictionaries with other nodes. To this end, we decompose a centralized quantization model into a set of decentralized sub-problems with consensus constraints and the alternating direction method of multipliers (ADMM) [12]. As a result, these sub-problems can be efficiently solved in parallel within a few iterations, and all the nodes obtain consistent quantizers learned from the distributed data. Since there is no exchange of training data across the nodes in the learning process, the communication cost of our DAMQ is low. Moreover, our approach can adapt to arbitrary network topologies.

Formally, we suppose the data is distributed across a set of P nodes in a network (*e.g.*, Figure 1). On the s-th node, there is a local set of N^s data points, denoted in matrix form as X^s. The global data $\mathrm{X} = \cup_{s=1}^{P} \mathrm{X}^s$ is then a concatenation of the local data matrix. When the data is distributed across the P nodes in an arbitrary network, the objective in Equation (13) can be rewritten as:

$$\begin{aligned} \min_{\mathrm{C,B}} \quad & \sum_{s=1}^{P} ||\mathrm{Y}_s - \mathrm{CB}_s||_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \mathrm{B} = \cup_{s=1}^{P} \mathrm{B}_s,\ \mathrm{B} = [\mathrm{b}_1^T \mathrm{b}_2^T \cdots \mathrm{b}_N^T], \\ & \mathrm{b}_n = [\mathrm{b}_{n1}^T \mathrm{b}_{n2}^T \cdots \mathrm{b}_{nM}^T]^T \\ & \mathrm{b}_{nm} \in \{0,1\}^K,\ ||\mathrm{b}_{nm}||_1 = 1, \\ & n = 1, \ldots, N, m = 1, \ldots, M. \end{aligned} \tag{17}$$

where B_s denotes the composition codes that belongs to the s-th node. We will further use C_s to denote the local copies of C on the s-th node.

4.1 Update C

When B is fixed, the dictionary C is shared across all nodes, which makes the problem hard to tackle. In order to make the objective separable, we enforce the consensus constraints $\mathrm{C}_s = \mathrm{C}_t$, for $\forall s, t \in \{1, 2, \ldots, P\}$ for $\{\mathrm{C}_s\}$ on all nodes, thus, we can transform Equation (17) to the following form without introducing any relaxation.

$$\begin{aligned} \min_{\{\mathrm{C}_s\}} \quad & \sum_{s=1}^{P} ||\mathrm{Y}_s - \mathrm{C}_s\mathrm{B}_s||_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \mathrm{C}_s = \mathrm{C}_t, \forall s, t \in \{1, 2, \ldots, P\}. \end{aligned} \tag{18}$$

The consensus constraint implies that all the local dictionaries should be consistent. Thanks to the transitivity between neighboring nodes in a connected graph, we are allowed to consider only the constraints between the neighboring nodes rather than all the constraints. For example, if the consensus constraints between all neighboring nodes are

satisfied in Figure 1, then $\mathrm{C}_3 = \mathrm{C}_6$ is naturally satisfied due to the fact that $\mathrm{C}_3 = \mathrm{C}_4 = \mathrm{C}_7 = \mathrm{C}_6$. Based on this observation, Equation (18) can be equivalently reformulated as:

$$\begin{aligned} \min_{\{\mathrm{C}_s\}} \quad & \sum_{s=1}^{P} ||\mathrm{Y}_s - \mathrm{C}_s\mathrm{B}_s||_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \mathrm{C}_s = \mathrm{C}_{s'}, s' \in \mathcal{N}(s), \forall s \in \{1, 2, \ldots, P\}. \end{aligned} \tag{19}$$

where $\mathcal{N}(s)$ represents the neighbors of the s-th node.

Next we show how the alternating direction method of multipliers (ADMM) [13] can be applied to decompose the global problem in (19) into several local subproblems.

4.1.1 Distributed Learning

ADMM is a variant of the augmented Lagrangian scheme that blends the decomposability of dual ascent with the method of multipliers. For our specific problem (19), the augmented Lagrangian is:

$$\begin{aligned} L(\mathrm{C}_s, \Lambda_{s,s'}) = & \sum_{s=1}^{P} ||\mathrm{Y}_s - \mathrm{C}_s\mathrm{B}_s||_{\mathrm{F}}^2 \\ & + \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} \mathrm{tr}\Big({\Lambda_{s,s'}}^T(\mathrm{C}_s - \mathrm{C}_{s'})\Big) \\ & + \frac{\rho}{2} \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} ||\mathrm{C}_s - \mathrm{C}_{s'}||_{\mathrm{F}}^2, \end{aligned} \tag{20}$$

where $\Lambda_{s,s'}$ is the Lagrangian multipliers corresponding to the constraints $\mathrm{C}_s = \mathrm{C}_{s'}, \forall s \in \{1, 2, \ldots, P\}$ and $s' \in \mathcal{N}(s)$. $\rho > 0$ is the penalty parameter of augmented Lagrangian. ADMM solves a problem of this form by repeating the following two steps [24]:

$$\begin{aligned} \mathrm{C}_s^{(k+1)} := & \arg\min_{\mathrm{C}^s} \sum_{s=1}^{P} ||\mathrm{Y}_s - \mathrm{C}_s\mathrm{B}^s||_{\mathrm{F}}^2 \\ & + \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} \mathrm{tr}\Big((\Lambda_{s,s'}^{(k)})^T(\mathrm{C}_s - \mathrm{C}_{s'}^{(k)})\Big) \\ & + \frac{\rho}{2} \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} ||\mathrm{C}_s - \mathrm{C}_{s'}^{(k)}||_{\mathrm{F}}^2 \end{aligned} \tag{21a}$$

$$\Lambda_{s,s'}^{(k+1)} := \Lambda_{s,s'}^{(k)} + \rho(\mathrm{C}_s^{(k+1)} - \mathrm{C}_{s'}^{(k+1)}). \tag{21b}$$

Despite the algorithm's elegance in form, the subproblems are still difficult to solve.

4.1.2 Simplification of Lagrange Multipliers

In the above update rule (21), assuming each node has t neighboring nodes in average in the network, about Pt Lagrange multipliers have been introduced into the optimization. Such a large number of multipliers remarkably enlarge the computation load of algorithm. However, due to the symmetry of an undirected graph, it is clear that if $s' \in \mathcal{N}(s)$ then $s \in \mathcal{N}(s')$. That is to say, every available constraint in the Equation (19) has been considered at least twice, *i.e.*, $\mathrm{C}_s = \mathrm{C}_{s'}$ and $\mathrm{C}_{s'} = \mathrm{C}_s$, which suggests that we can simplify the update rules in Equation (21). First of all, we can rewrite the second term of Equation (21a) in another form as:

$$\begin{aligned} & \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} \mathrm{tr}\Big({\Lambda_{s,s'}^{(k)}}^T(\mathrm{C}_s - \mathrm{C}_{s'}^{(k)})\Big) \\ & = \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} \mathrm{tr}\Big((\Lambda_{s,s'}^{(k)})^T\mathrm{C}_s\Big) - \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} \mathrm{tr}\Big((\Lambda_{s,s'}^{(k)})^T\mathrm{C}^{s'}\Big) \\ & = \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} \mathrm{tr}\Big((\Lambda_{s,s'}^{(k)})^T\mathrm{C}_s\Big) - \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} \mathrm{tr}\Big((\Lambda_{s',s}^{(k)})^T\mathrm{C}_s\Big) \\ & = \sum_{s=1}^{P} \mathrm{tr}\Big(\sum_{s' \in \mathcal{N}(s)} (\Lambda_{s,s'}^{(k)} - \Lambda_{s',s}^{(k)})^T\mathrm{C}_s\Big). \end{aligned} \tag{22}$$

In addition, owing to the symmetric characteristics, we can easily write down the symmetrical counterpart of Equation (21b) as follows:

$$\Lambda_{s',s}^{(k+1)} := \Lambda_{s',s}^{(k)} + \rho(\mathrm{C}_{s'}^{(k+1)} - \mathrm{C}_s^{(k+1)}). \tag{23}$$

For any two adjacent nodes, with Equation (21b) and (23), we have:

$$\Lambda_{s,s'}^{(k+1)} - \Lambda_{s',s}^{(k+1)} := (\Lambda_{s,s'}^{(k)} - \Lambda_{s',s}^{(k)}) + 2\rho(\mathrm{C}_s^{(k+1)} - \mathrm{C}_{s'}^{(k+1)}). \tag{24}$$

Therefore, by defining P new Lagrange Multipliers Λ_s as:

$$\Lambda_s = \sum_{s' \in \mathcal{N}(s)} (\Lambda_{s,s'} - \Lambda_{s',s}). \tag{25}$$

The update rule of ADMM in Equation (21) can be simplified as:

$$\begin{aligned} \mathrm{C}_s^{(k+1)} := & \arg\min_{\mathrm{C}_s} \sum_{s=1}^{P} ||\mathrm{Y}_s - \mathrm{C}_s\mathrm{B}_s||_{\mathrm{F}}^2 \\ & + \sum_{s=1}^{P} \mathrm{tr}\Big((\Lambda_s^{(k)})^T\mathrm{C}_s\Big) \\ & + \frac{\rho}{2} \sum_{s=1}^{P} \sum_{s' \in \mathcal{N}(s)} ||\mathrm{C}_s - \mathrm{C}_{s'}^{(k)}||_{\mathrm{F}}^2 \end{aligned} \tag{26a}$$

$$\Lambda_s^{(k+1)} := \Lambda_s^{(k)} + 2\rho(\mathrm{C}_s^{(k+1)} - \mathrm{C}_{s'}^{(k+1)}). \tag{26b}$$

Obviously, the updates of local variables C_s and Λ_s in Equation (26) can be separated into P subproblems, and thus can be carried out independently in parallel across the nodes.

4.1.3 Solution to subproblems

At last, we show how to solve the subproblem on each node. Formally, the s-th subproblem on the s-th node can be written as following:

$$\begin{aligned} \phi(\mathrm{C}_s; \mathrm{B}_s) = & ||\mathrm{Y}_s - \mathrm{C}_s\mathrm{B}_s||_{\mathrm{F}}^2 \\ & + \mathrm{tr}\Big((\Lambda_s^{(k)})^T\mathrm{C}^s\Big) + \frac{\rho}{2} \sum_{s' \in \mathcal{N}(s)} ||\mathrm{C}_s - \mathrm{C}_{s'}^{(k)}||_{\mathrm{F}}^2, \end{aligned} \tag{27}$$

which is an unconstrained quadratic problem with respect to C_s. The derivative with respect to C_s is as follows:

$$\begin{aligned} \frac{d}{d\mathrm{C}_s}\phi(\mathrm{C}_s; \mathrm{B}_s) = & -2\,(\mathrm{Y}_s - \mathrm{C}_s\mathrm{B}_s){\mathrm{B}_s}^T \\ & + \Lambda_s^{(k)} + \rho \sum_{s' \in \mathcal{N}(s)} (\mathrm{C}_s - \mathrm{C}_{s'}^{(k)}). \end{aligned} \tag{28}$$

By setting Equation (28) to 0, we obtain the following update rule for C_s as:

$$\mathrm{C}_s := \left[2\mathrm{Y}_s{\mathrm{B}_s}^T - \Lambda_s^{(k)} + \rho \sum_{s' \in \mathcal{N}(s)} \mathrm{C}_{s'}^{(k)}\right]\left[2\mathrm{B}_s{\mathrm{B}_s}^T + \rho t \mathrm{I}\right]^{-1}, \tag{29}$$

where t stands for the number of neighbors of the s-th node.

4.2 Update B

When C is fixed, each B_s can be locally updated in parallel on each node. For example, to update b_n, the codes for the n-th data vector y_n on the s-th node, the subproblem to tackle is as following:

$$\begin{aligned} \min_{\mathrm{b}_n} \quad & ||\mathrm{y}_n - \mathrm{C}_s\mathrm{b}_n||^2 \\ \text{s.t.} \quad & \mathrm{b}_n = [{\mathrm{b}_{n1}}^T {\mathrm{b}_{n2}}^T \cdots {\mathrm{b}_{nM}}^T]^T \\ & \mathrm{b}_{nm} \in \{0,1\}^K, ||\mathrm{b}_{nm}||_1 = 1 \\ & m = 1, 2, \ldots, M. \end{aligned} \tag{30}$$

We adopt the same iterative update method as Section 3.4.2 to solve this problem.

4.3 Analysis

We have presented the whole procedure of our proposed DAMQ. Note that the update of local C_s, B_s and Λ_s can all be conducted in parallel on each node, which is the key factor for our method to work in a distributed setting.

4.3.1 Analysis on Convergence

It is easy to verify that Equation (17) is lower-bounded (not smaller than 0), and the updates for B always decreases the value of Equation (17), hence, the convergence of DAMQ is conditioned on the convergence of ADMM for Equation (20). Clearly, Equation (20) is convex with respect to C, which the theoretical convergence property of the ADMM is guaranteed [12]. In practice, it takes around 15 iterations to converge as shown in Figure 2.

4.3.2 Analysis on Communication Complexity

Here we analyze the communication complexity of the proposed distributed composite quantization. Recall that M denotes the number of dictionaries, K denotes the number of elements in each dictionary, d denotes the dimension of data, and N_s is the number of local samples in the s-th node.

In our algorithm, each node only shares the dictionary C_s with its neighboring nodes. Supposing the s-th node has t neighbors, the communication complexity is $\mathcal{O}(tMKd)$, which is independent to N_s.

5 Experiments

We compare our AMQ with several state-of-the-art methods: Product Quantization (PQ) [20], Optimized Product Quantization (OPQ) [8], Cartesian k-means (CKM) [27], Composite Quantization (CQ) [40], Additive Quantization [1], iterated Local Search for AQ (LSQ) [25], Arborescence Coding (ArborC) [4] and MultiScale Quantization (MSQ) [37]. We evaluate these algorithms in ANN search and image retrieval task by performing linear scan search using asymmetric distance [20]. We note that our algorithm are also useful in other applications, such as distance/similarity learning [5], [30], feature map approximation [36], clustering [29] and cross-modal retrieval [23].

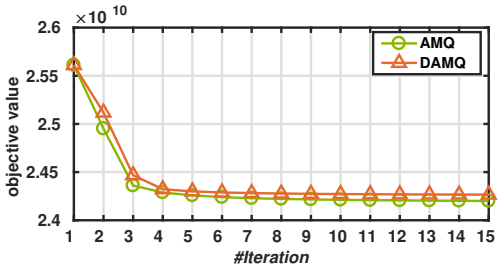

Fig. 2. Convergence of our AMQ and DAMQ on SIFT1B dataset with 64-bit codes.

To make comparisons between DCQ [13] and DAMQ, we randomly distribute the training data to different nodes in a network. We construct a network with 10 simulated nodes, as shown in Figure 1. Our machine is equipped with 24 Intel Xeon CPUs E5-2630 (2.30GHz) and 96 GB memory. We empirically set the penalty parameter = 100 and the number of ADMM iterations $\mathrm{I} = 5$. Ideally, the learned local dictionaries are supposed to be consistent with sufficiently large. In practice, we use the local dictionaries to perform linear scan search for both DCQ and DAMQ following [13].

5.1 Evaluation on ANN Search

We perform the ANN experiments on three datasets: SIFT1M [20], consisting of 1M 128-dimensional SIFT features as base vectors, 100K learning vectors and 10K queries; DEEP1M [3], containing 1M 256-dimensional PCA-projected deep features as base vectors, 500K learning vectors and 1K queries; and SIFT1B [22], composed of 1B SIFT features as base vectors, 100M learning vectors and 10K queries. Following the convention [1], [40], we measure the search quality by recall@R, *i.e.*, for varying R, the average rate of queries for which the 1-nearest neighbor is ranked in the top R positions. Similar to [14], [16], the ground-truth nearest neighbors are computed over the original features using linear scan.

Table 1 presents the comparison on SIFT1M and DEEP1M using 64-bit codes. One can see that partition-based methods such as PQ [20], OPQ [8] and CKM [27] performs not as well as addition-based methods like CQ [40] and LSQ [25], because they do not well exploit the data distribution during subspace partitioning. Our approach outperforms other methods except for recall@1 on SIFT1M dataset. Also, it can be observed that the performance of DAMQ is very close to that of AMQ, which suggests that learning composite quantizers in a distributed setting does not compromise much quality compared to the centralized version. Meanwhile, our DAMQ consistently outperforms DCQ, thanks to the superior performance of AMQ to CQ. Our method DAMQ even outperforms the centralized quantization algorithms CQ, which demonstrates the power of the proposed distributed learning scheme.

Table 2 shows the results of large scale datasets: SIFT1B. Similar to [40], we use the first 1M learning vectors for

TABLE 1
The recall@R for different algorithms on SIFT1M and DEEP1M datasets with 64-bit codes.

Method	SIFT1M recall@1	recall@2	recall@5	recall@10	recall@20	recall@50	Method	DEEP1M recall@1	recall@2	recall@5	recall@10	recall@20	recall@50
PQ [20]	22.53	32.34	46.99	60.14	72.03	83.49	PQ [20]	20.13	30.24	44.69	57.84	69.83	81.09
OPQ [8]	24.34	35.71	52.47	63.89	74.20	86.50	OPQ [8]	21.14	31.21	49.27	60.09	70.80	84.50
CQ [40]	29.48	42.97	58.40	71.09	82.41	93.02	CQ [40]	27.48	40.17	55.80	70.09	80.41	90.62
AQ [1]	31	-	-	-	-	-	AQ [1]	-	-	-	-	-	-
LSQ [25]	29.37	43.71	60.72	72.54	84.63	94.72	LSQ [25]	27.37	40.71	57.72	71.54	82.63	91.72
ArborC [4]	31.60	-	-	-	-	-	ArborC [4]	24.30	-	-	-	-	-
MSQ [37]	**32.77**	39.50	55.88	71.22	78.88	83.77	MSQ [37]	-	-	-	-	-	-
AMQ	32.15	**46.39**	**62.04**	**75.30**	**86.52**	**96.41**	**AMQ**	**28.37**	**42.59**	**60.38**	**73.62**	**84.28**	**94.17**
DCQ [13]	28.03	41.47	57.72	70.69	81.30	92.34	DCQ [13]	26.48	39.17	54.80	68.09	78.41	87.62
DAMQ	**31.17**	**45.39**	**61.88**	**73.22**	**85.88**	**94.79**	**DAMQ**	**27.64**	**41.28**	**58.84**	**71.63**	**81.45**	**91.50**

TABLE 2
The recall@R for different algorithms on SIFT1B datasets with 64-bit and 128-bit codes.

Method	SIFT1B, 64-bit codes recall@1	recall@2	recall@5	recall@10	recall@20	recall@50	Method	SIFT1B, 128-bit codes recall@1	recall@2	recall@5	recall@10	recall@20	recall@50
PQ [20]	6.81	10.47	17.90	24.58	33.12	45.70	PQ [20]	26.91	38.34	56.63	70.24	81.09	91.75
OPQ [8]	7.27	11.85	19.95	27.75	36.54	49.69	OPQ [8]	28.96	41.49	59.15	72.76	82.97	92.06
CKM [27]	8.95	12,72	21.48	30.54	39.80	53.03	CKM [27]	28.14	41.19	60.42	73.18	84.91	93.57
CQ [40]	9.79	13.22	23.95	33.53	45.19	59.67	CQ [40]	34.03	48.57	68.84	81.91	90.53	95.40
LSQ [25]	10.67	15.87	24.75	37.77	46.74	60.31	LSQ [25]	36.39	52.22	72.65	83.55	91.17	97.12
AMQ	**11.70**	**16.60**	**25.03**	**39.18**	**48.27**	**63.69**	**AMQ**	**38.04**	**57.62**	**75.09**	**85.71**	**95.82**	**98.35**
DCQ [13]	8.13	12.86	22.49	31.43	43.24	56.73	DCQ [13]	32.84	46.07	66.24	78.43	87.81	92.24
DAMQ	**9.24**	**14.35**	**24.92**	**35.77**	**46.67**	**60.41**	**DAMQ**	**35.21**	**53.47**	**71.33**	**82.35**	**91.18**	**95.26**

TABLE 3
The mAP on the Holidays dataset with distractors.

	#Bits	PQ [20]	OPQ [8]	CKM [27]	LSQ [25]	CQ [40]	**AMQ**	DCQ	**DAMQ**
Fisher	32	0.451	0.469	0.497	0.505	0.501	**0.512**	0.503	**0.508**
	64	0.471	0.492	0.538	0.564	0.560	**0.570**	0.554	**0.561**
	128	0.496	0.517	0.568	0.610	0.602	**0.625**	0.595	**0.607**
VLAD	32	0.484	0.493	0.506	0.512	0.508	**0.517**	0.506	**0.518**
	64	0.519	0.526	0.548	0.577	0.572	**0.581**	0.575	**0.580**
	128	0.538	0.562	0.576	0.619	0.614	**0.634**	0.610	**0.626**

efficient training. It can be seen that our approach consistently performs the best across different code lengths, and the curves of our DAMQ and AMQ almost overlap, which again validates the efficacy of our generalization of AMQ to the distributed settings. These results are consistent with the findings in Table 1.

5.2 Evaluation on Image Retrieval Application

We report the results of different quantization methods on image retrieval. The images are represented as an aggregation of local descriptors, often thousands of dimensions. We evaluate the performances over the 4096-dimensional Fisher vectors [28] and the 4096-dimensional VLAD vectors [21] extracted from the INRIA Holidays data set [19] that contains 500 query and 991 relevant images. Following common practice [19], we use extra one million MIRFlickr-1M images [18] as distractors.

The search performances in terms of mean Average Precision (mAP) [15], [35] are shown in Table 3. Our method performs the best thanks to its effective ANN search. The performance of our approach DAMQ is close to AMQ, overall better than DCQ, which is consistent with what we have observed in ANN search experiments in Section 5.1.

5.3 Evaluation on Efficiency

Although both DAMQ and AMQ require around 15 iterations to converge, DAMQ could largely reduce the training time by exploiting parallel computation, which is beneficial to large-scale datasets like videos [38], [39]. To quantitatively show the efficiency advantages of DAMQ, we evaluate the training time of DAMQ and AMQ on the SIFT1B dataset using different code lengths. We also vary the number of nodes to see how training time can be shortened with more nodes. When the same number of nodes are employed, the training time might be affected by the topology of the network. For convenience, we choose two representative topologies in this experiment, *i.e.*, binary tree topology and line topology.

Due to the lack of a real computing cluster, we measure the training time by simulating each node sequentially on a single machine. To be more specific, each simulated node solely occupies the entire machine in turn for training on its local data, *e.g.*, updating the local dictionaries, and the time cost for the slowest node to update its local dictionaries will be regarded as the training time for the simulated cluster to update all local dictionaries. Similar protocols are also applied to obtain the time for updating B. The communication time is neglected due to the small communication complexity as analyzed in Section 4.3.2. For instance, the dictionary size is only 2.09 MB with the code length as 128 bits ($M = 16, K = 8$) for SIFT1B dataset.

Figure 3 presents the training time under different settings. Roughly speaking, the training time seems to increase quadratically as the code length increases. Meanwhile, the more nodes involved in computation, the shorter the training time. This phenomenon is intuitive. With more nodes involved in the computation, the size of data distributed to each node becomes smaller, thus the corresponding training time is less. Take 128-bit code length as an example, the training time of DAMQ with 16-node binary tree topology is about 30.4 hours on the learning set of SIFT1B of 100M samples, while AMQ takes 82.8 hours using a single machine. Also, it could be observed that the line topology takes a few more minutes to converge than binary tree topology, but the total training time is still acceptable considering that

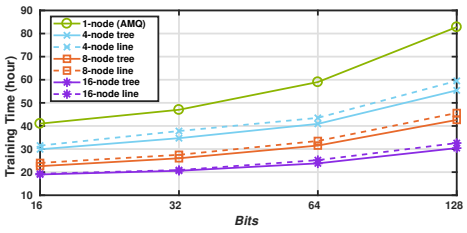

Fig. 3. The vertical axis stands for the training time (minute), the horizontal axis for different code lengths.

TABLE 4
The results are obtained on SIFT1B dataset using 64-bit codes.

# nodes / topology	1	4		8		16	
		tree	line	tree	line	tree	line
# iters to converge	12	13	15	11	12	14	17
objective value ($\times 10^{10}$)	2.4203	2.4259	2.4241	2.4197	2.42055	2.4271	2.4288
recall@100 (/%)	77.51	74.19	75.07	74.86	74.39	75.11	75.32

line topology is already the most undesirable case among all topologies. These results demonstrate the potentials of DAMQ for massive data in real-world applications.

Note that the number of iterations to convergence, the objective value at convergence, and the recall@R in testing phase are slightly different under different settings. We measure those 3 quantities on SIFT1B dataset using 64-bit code. As shown in Table 4, those 3 quantities do not vary a lot with the network topology or the number of nodes, which demonstrates the robustness of DAMQ. For the number of iterations to convergence, line topology takes more iterations than binary tree topology, probably due to the larger diameter of line topology. However, we do not observe correlations between the objective value at convergence and the recall@R in testing phase, and the network topology or the number of nodes.

6 Conclusion

In this paper, we present a compact coding approach, Asymmetric Mapping Quantization (AMQ), to approximate nearest neighbor search. The superior search accuracy stems from the novel formulation of AMQ, which avoids the computation of the norm of database vectors by an efficient inner product similarity via lookup tables. A distributed generalization of AMQ is proposed to accelerate the training on large-scale dataset. Empirical results on various tasks and datasets validate the merits of the proposed approach.

Acknowledgments

This work is supported in part by start-up funds from University at Buffalo.

References

[1] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *CVPR*, 2014.
[2] A. Babenko and V. Lempitsky. Tree quantization for large-scale similarity search and classification. In *CVPR*, 2015.
[3] A. Babenko and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *CVPR*, 2016.
[4] A. Babenko and V. Lempitsky. Annarbor: Approximate nearest neighbors using arborescence coding. In *ICCV*, 2017.
[5] S. Chang, G.-J. Qi, C. C. Aggarwal, J. Zhou, M. Wang, and T. S. Huang. Factorized similarity learning in networks. In *ICDM*, 2014.
[6] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Google's globally distributed database. *TOCS*, 2013.
[7] C. Du and J. Wang. Inner product similarity search using compositional codes. *arXiv preprint arXiv:1406.4966*, 2014.
[8] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *TPAMI*, 2014.
[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS operating systems review*. ACM, 2003.
[10] S. Greenhill and S. Venkatesh. Distributed query processing for mobile surveillance. In *ACM conf. on Multimedia*, 2007.
[11] J.-P. Heo, Z. Lin, and S.-E. Yoon. Distance encoded product quantization. In *CVPR*, 2014.
[12] M. R. Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 1969.
[13] W. Hong, J. Meng, and J. Yuan. Distributed composite quantization. In *AAAI*, 2018.
[14] W. Hong, J. Meng, and J. Yuan. Tensorized projection for high-dimensional binary embedding. In *AAAI*, 2018.
[15] W. Hong and J. Yuan. Fried binary embedding: From high-dimensional visual features to high-dimensional binary codes. In *TIP*, 2018.
[16] W. Hong, J. Yuan, and S. D. Bhattacharjee. Fried binary embedding for high-dimensional visual features. In *CVPR*, 2017.
[17] A. Howard, M. J. Matarić, and G. S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Distributed Autonomous Robotic Systems*. Springer, 2002.
[18] M. J. Huiskes and M. S. Lew. The mir flickr retrieval evaluation. In *ACM conf. on Multimedia Information Retrieval*, 2008.
[19] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, 2008.
[20] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 2011.
[21] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010.
[22] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: re-rank with source coding. In *ICASSP*, 2011.
[23] K. Li, G.-J. Qi, J. Ye, and K. A. Hua. Linear subspace ranking hashing for cross-modal retrieval. *TPAMI*, 2017.
[24] J. Liang, M. Zhang, X. Zeng, and G. Yu. Distributed dictionary learning for sparse representation in sensor networks. *TIP*, 2014.
[25] J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In *ECCV*, 2016.
[26] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 1980.
[27] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, 2013.
[28] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.
[29] G.-J. Qi, C. C. Aggarwal, and T. S. Huang. On clustering heterogeneous social media objects with outlier links. In *ACM conf. on Web Search and Data Mining*, 2012.
[30] G.-J. Qi, X.-S. Hua, and H.-J. Zhang. Learning semantic distance from community-tagged media collection. In *ACM conf. on Multimedia*, 2009.
[31] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2006.
[32] J. Wang and T. Zhang. Composite quantization. *CoRR*, 2017.
[33] J. Wang, T. Zhang, N. Sebe, H. T. Shen, et al. A survey on learning to hash. *TPAMI*, 2017.
[34] X. Wang, T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Supervised quantization for similarity search. In *CVPR*, 2016.
[35] Z. Wang and J. Yuan. Simultaneously discovering and localizing common objects in wild images. *TIP*, 2018.
[36] Z. Wang, X.-T. Yuan, Q. Liu, and S. Yan. Additive nearest neighbor feature maps. In *ICCV*, 2015.
[37] X. Wu, R. Guo, A. T. Suresh, S. Kumar, D. N. Holtmann-Rice, D. Simcha, and F. X. Yu. Multiscale quantization for fast similarity search. In *NIPS*, 2017.
[38] J. Ye, K. Li, G.-J. Qi, and K. A. Hua. Temporal order-preserving dynamic quantization for human action recognition from multimodal sensor streams. In *ACM conf. on Multimedia Retrieval*, 2015.
[39] J. Ye, G.-J. Qi, N. Zhuang, H. Hu, and K. A. Hua. Learning compact features for human activity recognition via probabilistic first-take-all. *TPAMI*, 2018.
[40] T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *ICML*, 2014.
[41] T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Sparse composite quantization. In *CVPR*, 2015.