# Competitive Quantization for Approximate Nearest Neighbor Search

Ezgi Can Ozan, Serkan Kiranyaz, *Senior Member, IEEE*, and Moncef Gabbouj, *Fellow, IEEE*

**Abstract**—In this study, we propose a novel vector quantization algorithm for Approximate Nearest Neighbor (ANN) search, based on a joint competitive learning strategy and hence called as competitive quantization (CompQ). CompQ is a hierarchical algorithm, which iteratively minimizes the quantization error by jointly optimizing the codebooks in each layer, using a gradient decent approach. An extensive set of experimental results and comparative evaluations show that CompQ outperforms the-state-of-the-art while retaining a comparable computational complexity.

**Index Terms**—Approximate nearest neighbor search, binary codes, large-scale retrieval, vector quantization

THE vast increase in size and dimension of today's datasets bring about new problems, as traditional methods fail to satisfy the present-day requirements. Like many others, the problem of fast and efficient distance calculations between pairs of samples leads researchers to approximate solutions. Binary embedding of descriptor vectors for faster distance calculations has become a highly popular research topic in recent years, drawing a significant attention [1]. The common approach is to encode the vectors as binary strings and compress very large datasets in much smaller sizes, decreasing the storage cost. Furthermore, the approximation of the distance between two vectors by using pre-calculated distance values gives a significant boost in terms of the search speed.

Binary embedding methods can be divided into two major branches as hashing and vector quantization. Hashing based approaches aim to approximate the distance between vectors using the Hamming distance [2], [3], [4], [5], [6]. These approaches are proved to be fast, as the Hamming distance calculation between two binary strings is basically an XOR operation, but since the Hamming distance is an integer between 0 and the length of the binary string, several vector pairs end up with the exact same distance approximation, which is a disadvantage in terms of retrieval rankings. Approaches such as [7], [8], [9] apply weighted Hamming distances, in order to add more variety to the results of the distance approximation by using look-up tables, which improves the performance; however, this slows down the search since distance approximations cannot be calculated simply by an XOR operation.

- *E.C. Ozan and M. Gabbouj are with the Department of Signal Processing, Tampere University of Technology, Tampere 33720, Finland. E-mail: {ezgi.ozan, moncef.gabbouj}@tut.fi.*
- *S. Kiranyaz is with the Electrical Engineering Department, College of Engineering, Qatar University, Doha 2713, Qatar. E-mail: mkiranyaz@qu.edu.qa.*

*Manuscript received 22 Feb. 2016; revised 29 June 2016; accepted 5 July 2016. Date of publication 10 Aug. 2016; date of current version 3 Oct. 2016. Recommended for acceptance by R. Cheng. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2016.2597834*

The introduction of weighted distances and look-up tables for hashing opened the doors for new binary embedding approaches, which constitutes the second major branch: the Vector Quantization (VQ). VQ is a very well-studied area in many fields such as electronics, telecommunication and signal processing. The application of VQ in binary embedding methods for approximate distance calculations, or as more formally stated in the literature, Approximate Nearest Neighbor (ANN) search starts with the Product Quantization (PQ) proposed in [10]. In this study, Jégou et al. propose a division among the dimensions of the vector. They perform quantization separately at each subspace, and obtain the final quantized vector as the Cartesian products of the sub-quantized vectors. This opens a new era in quantization for ANN as with this method, the number of quantization centers can scale up to very large numbers. Several improvements have been applied on top of PQ such as [11], [12], [13], [14], [15], obtaining significant increase in performance.

Besides PQ and its variants, for the purpose of scaling up the number of quantization codevectors exponentially, another approach is to quantize a given vector as the *addition* of several codevectors. Chen et al. in [16] propose this approach to ANN search problems. The authors introduce several layers of quantization, as each layer quantizes the residuals of the previous layer. Improvements on this method have been proposed in [17], [18]. In [19], Babenko et al. propose a method again based on quantization by addition of several codevectors, yet they remove the hierarchy between layers of residual quantization, providing a high level of freedom for the quantization codevectors. However, this freedom requires an encoding step, which is computationally very expensive, but still it outperforms the state-of-the-art methods. Other methods such as [20], [21], [22] add further constraints on the selection of codevectors, in order to obtain a more efficient encoding step.

In summary, many recent methods propose quantization through the addition of several codevectors, but they have to choose between using either highly constrained codevector proposals as in [16], which results in inferior performance; or leaving more freedom for codevectors while looking for a

heuristic search, which results in computationally expensive encoding. To address this drawback in an efficient way we propose a novel VQ approach, which outperforms the state-of-the-art quantization methods, with a comparable computational complexity. The novel contributions in this paper can be summarized as follows:

- Proposing a novel addition based quantization method, which preserves the layer hierarchy as in [16], consisting of a novel method which jointly trains all the codebooks, and minimizes the quantization error together in all layers.
- Proposing a better encoding approach by redefining *'the winner codevector'*, which provides lower quantization error.

The rest of the paper is organized as follows. In Section 1, the problem formulation is given together with a more detailed explanation of the related work. In Section 2, the proposed method is presented in detail. Section 3 presents the experimental results and comparisons with the state-of-the-art. In Section 0, the method and the obtained results are discussed thoroughly, and finally Section 5 concludes the paper.

## 1 Problem Formulation and Related Work

Vector quantization can be seen as an optimization problem, where the optimization criterion is to minimize the mean squared quantization error. The quantization error of a quantizer $Q$ can be described as follows: Given a set of $N$ vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, the mean squared quantization error $MSE_Q$ is defined as in (1).

$$MSE_Q = \frac{1}{N} \sum_{i}^{N} ||\boldsymbol{x}_i - Q(\boldsymbol{x}_i)||_2^2. \tag{1}$$

The quantizer $Q$ quantizes the given feature vector $\boldsymbol{x}_i$ to its corresponding codevector as,

$$Q\left(\boldsymbol{x}_i\right) = \boldsymbol{C}\boldsymbol{b}_i, \tag{2}$$

where $\boldsymbol{b}_i \in \{0,1\}^K$ is the binary selection vector, with $||\boldsymbol{b}_i||_1 = 1$. The main difference between traditional VQ and VQ for ANN is that, the number of codevectors for ANN is much greater than traditional vector quantization, because VQ for ANN targets large-scale datasets. As mentioned earlier, in order to increase the number of codevectors exponentially, VQ methods for ANN usually use several codebooks for quantization [1]. The use of $M$ codebooks increase the number of codevectors from $K$ to $K^M$. Codevectors from different codebooks are combined either by concatenation (Cartesian products) as in [10], [12], [13], [14], or by addition as in [16], [19], [20], [21], [22].

Vector quantization by Cartesian product of codevectors can be formulated as,

$$\widehat{\boldsymbol{x}}_i = \begin{bmatrix} \boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)} \\ \cdot \\ \cdot \\ \boldsymbol{C}^{(M)}\boldsymbol{b}_i^{(M)} \end{bmatrix} \qquad \boldsymbol{C}^{(m)}\boldsymbol{b}_i^{(m)} \in \mathbb{R}^{D/M} \tag{3}$$

where vector, $\hat{\boldsymbol{x}}$, is the quantization output of vector $\boldsymbol{x}$. $\boldsymbol{C}^{(m)} = \{\boldsymbol{c}_1^{(m)} \ldots \boldsymbol{c}^{(m)}{}_K\} \in \mathbb{R}^{D/M \times K}$ is the corresponding codebook of the $m$th subspace, $m \in \{1 \ldots M\}$, where $M$ is the number of codebooks (one codebook per subspace) and $K$ is the number of codevectors per codebook. $\boldsymbol{B}^{(m)} = \{\boldsymbol{b}_1^{(m)} \ldots \boldsymbol{b}_N^{(m)}\} \in \mathbb{R}^{K \times N}$ is the set of K-dimensional, binary codevector selection vectors for the $m$th codebook, where $\boldsymbol{b}_i^{(m)} \in \{0,1\}^K$ with $\boldsymbol{b}_i^{(m)}{}_1 = 1$.

Similarly, vector quantization by addition of codevectors can be formulated as given in (4). The main difference from (3) is that here all codebooks come from the original feature space, i.e., $\boldsymbol{C}_m = \{\boldsymbol{c}_{m1} \ldots \boldsymbol{c}_{mK}\} \in \mathbb{R}^{D \times K}$. Note that in this paper, codebooks obtained from the same feature space are represented using a subscript ($\boldsymbol{C}_m$), while codebooks belonging to different subspaces are represented using a superscript in parenthesis ($\boldsymbol{C}^{(m)}$). The corresponding binary selection vectors are also represented accordingly.

$$\widehat{\boldsymbol{x}}_i = \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{mi} \quad \boldsymbol{C}_m \boldsymbol{b}_{mi} \in \mathbb{R}^D. \tag{4}$$

Different codevectors obtained from different codebooks are combined in order to obtain $\hat{x}$, the final approximation of vector $x$, providing the minimum quantization error. This optimization problem can be formulated for the Cartesian product based VQ as given in (5), and for the addition based VQ as given in (6).

$$MSE_Q^{(Cartesian)} = \min_{\{\boldsymbol{C}^{(m)}\},\{\boldsymbol{B}^{(m)}\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x_i} - \begin{bmatrix} \boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)} \\ \cdot \\ \cdot \\ \boldsymbol{C}^{(M)}\boldsymbol{b}_i^{(M)} \end{bmatrix} \right\|_2^2 \tag{5}$$

$$MSE_Q^{(Addition)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{mi} \right\|_2^2. \tag{6}$$

Closed form solutions for the above problems do not exist, hence the recent methods approach this minimization using approximate techniques with different constraints and different heuristics. Constraints limit the search space and heuristics enable a more feasible and divergent search. In the next section, the state-of-the-art methods will be examined in terms of such constraints and the heuristics they define.

### 1.1 Product Quantization

Product Quantization [10] proposed by Jegou et al. is a Cartesian product based VQ method. It uses subspaces of the feature space to create different layers of quantization. On each subspace, a different codebook is trained separately. In other words, PQ divides the original feature space into $M$ subspaces, and each codevector $\boldsymbol{c}_k^{(m)} \in \mathbb{R}^{D/M}$ obtained on this subspace is a subvector. This approach splits the optimization problem into $M$ independent problems. However, the assumption of subspaces being statistically independent does not usually hold in practice, and many variants of PQ

have been proposed [11], [12], [13], [14] in order to overcome this drawback. The optimization problem of PQ can be formulated as given in (7), where $\boldsymbol{x}_i^{(m)}$ is the subvector of $\boldsymbol{x}_i$ corresponding to the $m$th subspace.

$$\begin{aligned} MSE_Q^{(PQ_1)} &= \min_{\{\boldsymbol{C}^{(1)}\},\{\boldsymbol{B}^{(1)}\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}_i^{(1)} - \boldsymbol{C}^{(1)} \boldsymbol{b}_i^{(1)} \right\|_2^2 \\ &\vdots \\ MSE_Q^{(PQ_M)} &= \min_{\{\boldsymbol{C}^{(M)}\},\{\boldsymbol{B}^{(M)}\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}_i^{(M)} - \boldsymbol{C}^{(M)} \boldsymbol{b}_i^{(M)} \right\|_2^2 \end{aligned} \tag{7}$$

### 1.2 K-Subspace Quantization

In Cartesian product based approaches, the creation of subspaces is problematic because of the unrealistic assumption that subspaces are statistically independent. The proposed solution to this problem is a PCA transformation [11], [12], [13], [14], which brings another problem of unbalanced distribution of information among dimensions. In [15], Ozan et al. also target this problem and propose to introduce more than one affine subspace, train a Transform Coding [11] variant in each and choose the best among them to quantize the samples. Introduction of multiple affine subspaces improves the assumption of independent subspaces after PCA. Besides, they propose to update codebooks in an iterative way, in order to minimize the quantization error further and obtain the state-of-the-art performance. The optimization problem for KSSQ can be formulated as given in (8). $\boldsymbol{X}_m \subset \boldsymbol{X}$ is a subset of samples such that $\cup_{m=1}^{M} \boldsymbol{X}_m = \boldsymbol{X}$ and $\boldsymbol{X}_m \cap \boldsymbol{X}_j = \emptyset$ where $m \neq j$. $\mathrm{L}_m$ is the number of dimensions for the $m$th subspace.

$$\begin{aligned} MSE_Q^{(KSSQ)} = \min_{\left\{\boldsymbol{C}_m^{(\mathrm{l}_m)}\right\},\left\{\boldsymbol{B}_m^{(\mathrm{l}_m)}\right\},\{X_m\},\{\mathrm{L}_m\}} \frac{1}{N} \sum_{m=1}^{M} \sum_{x_i \in X_m} \Bigg\| \boldsymbol{x}_i \\ - \begin{bmatrix} \boldsymbol{C}_m^{(1)} \boldsymbol{b}_{m_i}^{(1)} \\ \vdots \\ \boldsymbol{C}_m^{(\mathrm{L}_m)} \boldsymbol{b}_{m_i}^{(\mathrm{L}_m)} \end{bmatrix} \Bigg\|_2^2 \end{aligned} \tag{8}$$

### 1.3 Residual Vector Quantization

Residual Vector Quantization (RVQ) [16] proposed by Chen et al. is an addition based VQ method, which dictates a hierarchy among the codebooks. As the name suggests, each succeeding codebook is trained on the residuals of the previous layer. The optimization problem is separated into $M$ sub-problems, which are solved consecutively. The optimization problem of RVQ can be formulated as,

$$\begin{aligned} MSE_Q^{(RVQ_1)} &= \min_{\{\boldsymbol{C}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \left( \boldsymbol{x}_i - \sum_{m=1}^{1} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|_2^2 \\ &\vdots \\ MSE_Q^{(RVQ_M)} &= \min_{\{\boldsymbol{C}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \left( \boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|_2^2 \end{aligned} \tag{9}$$

This hierarchy simplifies the encoding process, as each codevector depends on the selection of the previous codevectors. However, the contribution of each layer to the minimization of the quantization error is not the same, as the last layers contribute much less than the first ones. This algorithm has been extended by several other methods [17], [18].

### 1.4 Optimized Cartesian K-Means

Optimized Cartesian K-Means (OCKM) [20] proposed by Wang et al. is an improvement over the Cartesian K-Means (CKM) [13], which forms the quantization vector as a Cartesian product of subvectors obtained from subspaces of the original feature space. Similar to CKM, OCKM proposes a multiplication with a rotation matrix $\boldsymbol{R}$ before dividing the initial vector space into subspaces. OCKM is a transition between the Cartesian product based vector quantization algorithms and the addition based vector quantization algorithms. Wang et al. propose to apply the same rotation and subspace division in CKM, but instead of picking one codevector per subspace, the authors suggest training two codebooks per subspace, hence picking two codevectors and adding them up. The optimization performed by OCKM can be formulated as

$$MSE_Q^{(OCKM)} = \min_{\left\{\boldsymbol{C}_c^{(m)}\right\},\left\{\boldsymbol{B}_c^{(m)}\right\},\{\boldsymbol{R}\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \boldsymbol{R} \begin{bmatrix} \sum_{c=1}^{2} \boldsymbol{C}_c^{(1)} \boldsymbol{b}_{c_i}^{(1)} \\ \vdots \\ \sum_{c=1}^{2} \boldsymbol{C}_c^{(\mathrm{M}/2)} \boldsymbol{b}_{c_i}^{(\mathrm{M}/2)} \end{bmatrix} \right\|_2^2 \tag{10}$$

OCKM generates the codebooks in different subspaces, yet within each subspace, there is no constraint on the codebooks and the codevectors they contain. In order to avoid the complexity of this unconstrained situation, they opt for more subspaces rather than codebooks i.e., they keep the number of codebooks per subspace limited to 2.

### 1.5 Additive Quantization

Additive Quantization (AQ) [19] is an unconstrained approach to addition based VQ. Babenko et al. propose to generate the quantized vectors using the addition of several different codevectors, each from a different codebook, trained without any constraints on the original feature space. The lack of constraints provides a boost in the quantization performance, yet the training and more importantly encoding procedure is extremely costly, as they propose to use *"heuristic beam search"* in the encoding process. Since there are no specific constraints, the optimization problem can be represented as in (6).

The complexity of the proposed beam search in [19] is proportional to the cubic order of the number of codebooks, the authors propose a PQ variant of the additive quantization, Additive Product Quantization (APQ) [19], which divides the feature space into two subspaces and performs AQ in each subspace independently, then concatenates both vectors to obtain the final quantized vector. This splits the optimization problem into two independent problems similar to PQ. The optimization problem for APQ can be formulated as given in (11)

$$MSE_Q^{(APQ_1)} = \min_{\{C_m^{(1)}\},\{B_m^{(1)}\}} \frac{1}{N}\sum_{i=1}^{N}\left\| {x^{(1)}}_i - \sum_{m=1}^{M/2} C_m^{(1)} b_{m_i}^{(1)} \right\|_2^2 .$$
$$MSE_Q^{(APQ_2)} = \min_{\{C_m^{(2)}\},\{B_m^{(2)}\}} \frac{1}{N}\sum_{i=1}^{N}\left\| {x^{(2)}}_i - \sum_{m=1}^{M/2} C_m^{(2)} b_{m_i}^{(2)} \right\|_2^2 . \quad (11)$$

### 1.6 Composite Quantization

Composite Quantization (CQ) [21] proposed by Zhang et al. is yet another addition based vector quantization method with constraints. CQ differs from the previous methods by the purpose of the selected constraints. Compared to the Cartesian product based methods, one serious drawback of the addition based VQ techniques is the computational cost of asymmetric distance calculation. Usually, Cartesian product based methods can calculate the approximate distance between an encoded database element and a given query vector in $M$ look-ups and additions, i.e., one look-up and addition for each codebook [10], [12], [13]. However, the computational cost for addition based methods has usually the complexity of $O(M^2)$ [16], [17], [19]. Zhang et al. propose bringing additional constraints in the codebook generation process so that the sum of the dot products of all codevectors from two different codebooks is equal to a constant value, i.e., $\sum_{m=1}^{M}\sum_{l=1,\, l\neq m}^{M}\left(C_m b_{m_i}\right)^T C_l b_{l_i} = \epsilon$. They include this constraint into the optimization via a penalty factor. The formulation of the optimization problem can be given as in (12), where $\mu$ is the penalty parameter

$$MSE_Q^{(CQ)} = \min_{\{C_m\},\{B_m\},\{\epsilon\}} \frac{1}{N}\sum_{i=1}^{N}\left\| x_i - \sum_{m=1}^{M} C_m b_{m_i} \right\|_2^2 + \mu \sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{l=1,\, l\neq m}^{M}\left( (C_m b_{m_i})^T C_l b_{l_i} - \epsilon \right)^2 . \quad (12)$$

### 1.7 (Optimized) Tree Quantization

Babenko et al. extended their previous work AQ, by adding more constraints on the codebooks and a rotation as in OPQ for the feature space, resulting with (Optimized) Tree Quantization (OTQ) [22]. In OTQ, a tree structure is introduced where each vertex of the tree is a codebook. Each dimension of the feature space is assigned to an edge in the tree, so each dimension is coded by two codebooks. It is assumed that any dimension that is not in the edge of a codebook is equal to zero. This brings orthogonality between the codevectors of any two codebooks that are not adjacent in the tree. The introduced tree structure significantly decreased the encoding complexity, while obtaining a comparable quantization performance with AQ. The optimization problem for OTQ can be formulated by (13). Here $[d]$ represents the $d$th dimension of the feature space and $a\,(m,n) = 1$ if dimension $d$ is assigned to edge $(m,n)$.

$$MSE_Q^{(OTQ)} = \min_{\{C_m\},\{B_m\},\{a(m,n)\}} \sum_{i=1}^{N}\sum_{d=1}^{D}\sum_{m=1}^{M}\sum_{n=m}^{M} a(m,n) \times \left| C_m b_{m_i}[d] + C_n b_{n_i}[d] - x_i[d] \right|^2 . \quad (13)$$

A noteworthy observation from the aforementioned prior studies in this domain is that the methods with less constraints offer a better quantization performance, whereas the complexity of the encoding and distance calculation increases. The methods with stronger constraints are faster yet perform worse. In this study, we aim to achieve a superior performance without increasing the computational complexity. In order to accomplish this we start from a well-constrained quantization with a hierarchical formation and then we shall relax these constraints for a better quantization performance whilst having a comparable computational complexity.

## 2 The Proposed Method: Competitive Quantization

As we discussed in the previous section, constraint selection is crucial as it significantly effects the quantization performance, computational complexity and search speed. In this paper, we focus on the hierarchical structure imposed by RVQ. The biggest advantage of RVQ's hierarchical approach is that, the codebooks have a given order, so encoding is simply selecting the nearest codevector in the current layer, calculating the residual and proceeding to the next layer. However, each codebook is trained separately, independent from the others, resulting in a suboptimal solution. In other words, the codebook training in the upper layers does not take the quantization error produced by the lower layers into account. In this study, we propose a joint optimization scheme updating all layers at the same time.

### 2.1 Competitive Codebook Learning

For addition based hierarchically connected quantization methods, for a given vector $\boldsymbol{x}$, a residual vector $\boldsymbol{r}_m$ at the $m$th level can be represented as given in (14).

$$\boldsymbol{r}_m = \boldsymbol{x} - \sum_{l=1}^{m-1} C_l b_l . \quad (14)$$

"Com*petitive Learning*" with a *winner-takes-all* strategy is one way to obtain the codebooks for such a connectionist quantization scheme [23]. In this scheme, each layer responds to its corresponding input by determining a winner codevector. The winner codevectors are updated and moved towards the input to minimize the error. As (14) shows, all the codevectors from all the levels are responsible from the obtained quantization error, meaning that all the codevectors should be updated together accordingly, to minimize this error.

In this paper, the codebooks are jointly optimized using the stochastic gradient decent, following the aforementioned competitive learning approach. With the stochastic gradient decent method, the codevectors are updated using the formula in (15).

$$\dot{\boldsymbol{c}}_m\,(t+1) = \dot{\boldsymbol{c}}_m(t) - \gamma_m(t)\nabla_{\dot{c}_m}\left( \left\| \boldsymbol{x} - \sum_{l=1}^{M} \dot{\boldsymbol{c}}_l \right\|_2^2 \right) \quad (15)$$

Here, the parameter $\gamma_m(t)$ is the learning rate at iteration $t$, which is decreased with every iteration. $\dot{\boldsymbol{c}}_m$ stands for the winner codevector for the sample vector $\boldsymbol{x}$ at the $m$th layer,

TABLE 1
CompQ Training Algorithm

INPUT: Training set $\boldsymbol{X} = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}\}$
OUTPUT: Quantization codebooks $\boldsymbol{C_m}$

- Initialize the codebooks.
- *for* $\boldsymbol{N_{it}}$ iterations
  - *for* each sample $\boldsymbol{x_i}$
    - Encode $\boldsymbol{x_i}$ in order to obtain the winner codevectors as given in **Section 2.3**
    - Calculate the quantization error vector $\boldsymbol{x_i} - \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m$
    - Update the winner codevectors with the error vector according to the equation in (18).
  - Decrease the learning rate by 1%.
- Return the quantization codebooks.

i.e., $\dot{c}_m = C_m b_m$ or equivalently;

$$\dot{\boldsymbol{c}}_m = \underset{c_{m_k}}{\operatorname{argmin}} \left\| \left( \boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right) - \boldsymbol{c}_{m_k} \right\|_2^2. \tag{16}$$

According to (16) the gradient of the error for the sample vector $\boldsymbol{x}$ can be calculated as,

$$\nabla_{\dot{c}_m} \left( \left\| \boldsymbol{x} - \sum_{l=1}^{M} \dot{\boldsymbol{c}}_l \right\|_2^2 \right) = 2 \left( \sum_{l=1}^{M} \dot{\boldsymbol{c}}_l - \boldsymbol{x} \right). \tag{17}$$

So, for each winner codevector at each level, the update rule, which is going to move the corresponding codevector towards the input can be formulated as,

$$\dot{\boldsymbol{c}}_m\,(t+1) = \dot{\boldsymbol{c}}_m(t) + 2\gamma_m(t) \left( x - \sum_{l=1}^{M} \dot{c}_l \right). \tag{18}$$

As the equation in (18) shows, at each iteration and for each sample, the winner codevectors in each layer should be updated by multiplying the learning rate with the error vector in order to minimize the total quantization error, as the result of the stochastic gradient descent approach. Note that, unlike RVQ, the codevectors from all layers are updated jointly. This prevents overfitting in upper levels and increases the contribution of lower levels to the minimization of the quantization error. The training algorithm including the iterative codevector updates is presented in Table 1.

### 2.2 Initialization of Codebooks

CompQ starts with a broad initialization of the codebooks. We use Transform Coding [11] in order to generate the initial codebooks for each layer. Transform Coding (TC) is also a VQ method for ANN. Using PCA, TC transforms the feature space into a new one, in which the dimensions are orthogonal. Centroids are then determined in the transform domain for each dimension. Combining these centroids yields the codevectors. The number of centroids for each dimension is proportional to the variance of the corresponding dimension, providing a balanced distribution of codevectors in the feature space. Initially, coarsely computed codebooks are selected in order to prevent early overfitting. We follow the hierarchy as formulated in (9). We train the TC for a layer, and obtain the initial codevectors, then calculate the residual vectors and proceed to the next layer.

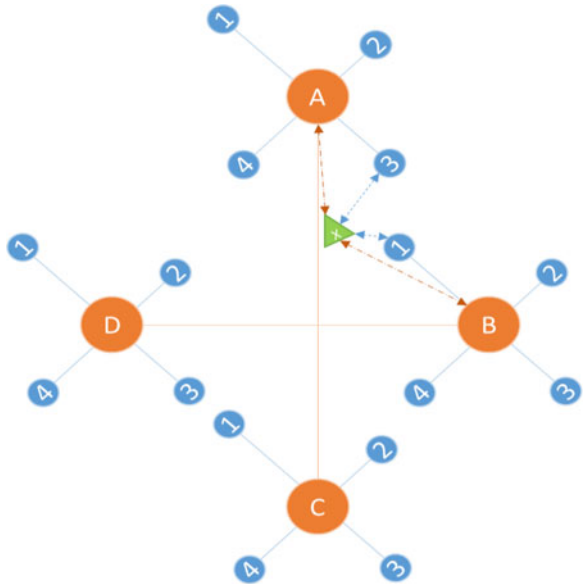


Fig. 1. Illustration of inferior encoding using hierarchical layers. The sample (X) is encoded with (A3) instead of the nearest code (B1).

### 2.3 Sample Encoding and Winner Codevector

Encoding of a new sample consists of finding out the winner codevector for each layer. The winner codevector $\dot{\boldsymbol{c}}_m$ can be defined as,

$$\dot{\boldsymbol{c}}_m = \underset{c_{m_k}}{\operatorname{argmin}} \left\| \left( \boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right) - \boldsymbol{c}_{m_k} \right\|_2^2. \tag{19}$$

Recall that in hierarchical structure based approaches such as RVQ and its variants, for each layer, the nearest codevector on the corresponding layer is chosen as the winner codevector. However, this may not be the best choice as it may lead to higher quantization errors. A toy example for the aforementioned problem is presented in Fig. 1. In this figure, the given sample (X) is encoded with (A3), since it is closer to the (A) in the upper layer than (B). However, (B1) would have been a better choice, as it is the nearest code to the given sample.

CompQ follows an alternative approach for the determination of the winner codevector. For each layer, instead of one codevector, the best $H$ candidates are picked and then the residual for each candidate is calculated. Then among $HK$ residuals, again the best $H$ are stored for the next layer. When the final layer is reached the winner codevectors for each layer have been obtained. Going back to Fig. 1, if (A) and (B) were considered as candidate codevectors for the first layer, then in the second layer (B1) would eventually be selected, as it gives the minimum quantization error. Note that, this is a special case of the beam search algorithm used in the encoding step of AQ [19]. A very similar approach is also proposed in OCKM [20]. In this paper, the hierarchical structure is imposed on the beam search and it limits the search space, hence reduces the computational complexity

TABLE 2
Test Results for SIFT1M, 32-bit Codes

| | recall@1 | recall@10 | recall@100 |
|---|---|---|---|
| *RVQ* | *NA* | *NA* | *NA* |
| *CKM/OPQ* | 0.068 | 0.273 | 0.658 |
| *AQ* | 0.106 | 0.415 | **0.825** |
| *CQ* | *NA* | *NA* | *NA* |
| *OCK* | *NA* | 0.348 | 0.742 |
| *ERVQ* | *NA* | *NA* | *NA* |
| *OTQ* | 0.093 | 0.368 | 0.793 |
| *KSSQ* | **0.145** | 0.434 | 0.802 |
| *CompQ* | 0.135 | **0.435** | 0.818 |

TABLE 3
Test Results for GIST1M, 32-bit Codes

| | recall@1 | recall@10 | recall@100 |
|---|---|---|---|
| *RVQ* | *NA* | *NA* | *NA* |
| *CKM/OPQ* | 0.054 | 0.142 | 0.396 |
| *AQ* | 0.069 | 0.189 | 0.467 |
| *CQ* | *NA* | *NA* | *NA* |
| *OCK* | *NA* | 0.172 | 0.467 |
| *ERVQ* | *NA* | *NA* | *NA* |
| *OTQ* | *NA* | *NA* | *NA* |
| *KSSQ* | **0.078** | 0.191 | 0.437 |
| *CompQ* | 0.072 | **0.200** | **0.504** |

drastically. Since AQ is not inclusive of any structure, the beam search in AQ searches for the best $H$ among $(M-m+1)HK$ codevectors for the $m$th layer. Thanks to the hierarchy in our case there are always $HK$ vectors to compare.

The distance calculation between a codevector and a residual is formulated in (20). After some mathematical manipulations, this equation can be rewritten to enable the use of look-up tables, and accelerate the encoding, as given in (21) and (22).

$$d\left(\boldsymbol{r}_{mi}, \boldsymbol{c}_{m,k}\right) = \left( \left\| \left( \boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l - \boldsymbol{c}_{m,k} \right) \right\|_2^2 \right) \tag{20}$$

$$d\left(\boldsymbol{r}_{m_i}, \boldsymbol{c}_{m,k}\right) = \left( \left\| \boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right\|_2^2 - 2\langle \boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l, \boldsymbol{c}_{m,k} \rangle + \left\| \boldsymbol{c}_{m,k} \right\|_2^2 \right) \tag{21}$$

$$d\left(\boldsymbol{r}_{m_i}, \boldsymbol{c}_{m,k}\right) = \left( \left\| \boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right\|_2^2 - 2\langle \boldsymbol{x}_i, \boldsymbol{c}_{m,k} \rangle + 2 \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l, \boldsymbol{c}_{m,k} + \left\| \boldsymbol{c}_{m,k} \right\|_2^2 \right). \tag{22}$$

Note that in (22), the first term is already calculated in the previous layer for all H candidates. The third and fourth term can be obtained from a look-up table. Only the second term should be calculated for each encoding operation.

### 2.4 Asymmetric Distance Calculation

The Asymmetric Distance [9], [24] is calculated between an encoded database element and a given query vector. For simplicity we omit the square root and represent the formulation of the square of the asymmetric distance, as given below:

$$d(\boldsymbol{x}, \bar{\boldsymbol{x}})^2 = \| x - \bar{\boldsymbol{x}} \|_2^2 = \left\| \boldsymbol{x} - \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m \right\|_2^2 \tag{23}$$

$$d(\boldsymbol{x}, \bar{\boldsymbol{x}})^2 = \|\boldsymbol{x}\|_2^2 - 2\langle \boldsymbol{x}, \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m \rangle + \left\| \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m \right\|_2^2 \tag{24}$$

$$d(\boldsymbol{x}, \bar{\boldsymbol{x}})^2 = \boldsymbol{x}_2^2 - 2 \sum_{m=1}^{M} \langle \boldsymbol{x}, \dot{\boldsymbol{c}}_m \rangle + \sum_{m=1}^{M} \sum_{l=1}^{M} \langle \dot{\boldsymbol{c}}_m, \dot{\boldsymbol{c}}_l \rangle. \tag{25}$$

As shown in (25), two look-up tables must be prepared for distance calculation beforehand. The first look-up table consists of dot-products of the query vector $\boldsymbol{x}$ with all codevectors. The second look-up table stores the dot-products calculated between pairs of codevectors. Since the first term of (25) is the same for all database elements, it can be neglected. The distance can be then calculated by $M + M^2$ look-ups and additions as in [16], [17], [18], [19].

## 3 Experimental Results

### 3.1 Exhaustive Search

Our approach is first tested on two publicly available datasets of 1 Million samples, SIFT1M and GIST1M [10] for exhaustive search. SIFT1M consists of 128-dimensional SIFT vectors and GIST1M consists of 960-dimensional GIST vectors.

We train our method using the given training sets and perform exhaustive search on both datasets for all given queries. We use $K = 256$, $H = 32$, $M = 8$ for 64-bits and $M = 4$ for 32-bits coding. The performance of CompQ is compared against the recent state-of-the-art methods such as, *Residual Vector Quantization* [16], *Cartesian K-Means/Optimized Product Quantization* (*CKM/OPQ*) [12], [13], *Additive Quantization* (*AQ/APQ*) [19], *Composite Quantization* [21], *Extended Residual Vector Quantization* (*ERVQ*) [17], *Optimized Cartesian K-Means* (*OCK*) [20] and *Optimized Tree Quantization* [22]. We do not compare against *Transform Coding* [11], *Product Quantization* [10], *Projected Residual Vector Quantization* [18] and *Distance Encoded Product Quantization* [25] since their results were already outperformed by the other compared methods. The results of the competing methods are taken from the original publications. For AQ/APQ, AQ is used for 32-bits coding and APQ for 64-bits as suggested by the authors. *NA* corresponds to missing results in the original publications.

The **recall@**$R$ measure is used for the experiments, which is the recall value for the first $R$ samples in retrieval. The nearest sample in the test set is taken as the ground truth for each query. We present the results for **recall@**1, **recall@**10 and **recall@**100 for 32-bit coding in Tables 2 and 3 and for 64-bit coding in Tables 4 and 5, respectively.

As observed from the results, the proposed method, *CompQ*, outperforms all recent state-of-the-art methods for all scores, on both datasets for 64-bits encoding. For 32-bits encoding, *CompQ* still has the best performance for recall@10 for SIFT1M and also for recall@10 and recall@100

TABLE 4
Test Results for SIFT1M, 64-bit Codes

| | recall@1 | recall@10 | recall@100 |
|---|---|---|---|
| *RVQ* | 0.257 | 0.659 | 0.952 |
| *CKM/OPQ* | 0.243 | 0.638 | 0.940 |
| *APQ* | 0.298 | 0.741 | 0.972 |
| *CQ* | 0.288 | 0.716 | 0.967 |
| *OCK* | 0.274 | 0.680 | 0.945 |
| *ERVQ* | 0.276 | 0.694 | 0.962 |
| *OTQ* | 0.317 | 0.748 | 0.972 |
| *KSSQ* | 0.325 | 0.754 | 0.976 |
| *CompQ* | **0.352** | **0.795** | **0.987** |

for GIST1M. *KSSQ* gives slightly better results than *CompQ* for recall@1 in both datasets for 32-bits encoding.

### 3.2 Non-Exhaustive Search

As mentioned above in Section 2.4, the asymmetric distance calculation requires $M + M^2$ look-ups and additions as in many addition based methods such as [16], [17], [18], [19]. This means the exhaustive search using those methods takes longer time compared to the Cartesian product based methods, which generally requires $M$ look-ups and additions. Methods such as CQ and OTQ propose additional constraints on codebooks to decrease this cost. In our method, we propose non-exhaustive search as a solution to this problem.

Non-exhaustive versions of methods such as PQ, OPQ have been implemented and tested using an additional coarse quantization layer as in [10], [12], [14], [26], [27]. However, in our method, a non-exhaustive search scheme can be implemented using the hierarchical structure. Since at each layer, residuals of the previous layers are quantized, upper layers can be used as coarse quantization layers and an inverted file list can be created, i.e., no additional coarse quantizer required.

Here we should state that, other non-exhaustive search algorithms such as [14], [26], [27], [28], [29] can also be applied on top of the proposed method, where the proposed method can be used as a subquantizer or for re-ranking purposes. Here we only discuss the non-exhaustive implementation of the proposed method, as this is an inherent property of it. We do not aim to propose a new indexing algorithm, as it would be beyond the scope of this paper.

A non-exhaustive version of RVQ (IVFRVQ) has been also proposed in [16], which uses the first $L$ layers as inverted file indexes. The query vector is compared to $K^L$ codevectors and the nearest $W$ codevectors are selected ($W < K$). In our method, we propose a minor modification to decrease this initial comparison overhead. The query vector is compared to the codevectors of the initial layer and the best $W$ of them are selected, and residuals for the second layer are calculated. Then among $KW$ residuals, the nearest $W^2$ are selected as target inverted file indexes, while in [16], the query is compared to all codevectors in the first two layers, resulting in $K^2$ comparisons. The comparison of the proposed non-exhaustive search algorithm with different $W$ values is given in Table 6.

TABLE 5
Test Results for GIST1M, 64-bit Codes

| | recall@1 | recall@10 | recall@100 |
|---|---|---|---|
| *RVQ* | 0.113 | 0.325 | 0.676 |
| *CKM/OPQ* | 0.118 | 0.334 | 0.715 |
| *AQ/APQ* | *NA* | *NA* | *NA* |
| *CQ* | 0.135 | 0.377 | 0.729 |
| *OCK* | 0.130 | 0.358 | 0.720 |
| *ERVQ* | 0.115 | 0.341 | 0.711 |
| *OTQ* | *NA* | *NA* | *NA* |
| *KSSQ* | 0.136 | 0.396 | 0.741 |
| *CompQ* | **0.155** | **0.419** | **0.801** |

TABLE 6
Non-Exhaustive Search, SIFT1M, $L = 2$, 64-bit Codes

| | Avg. No. Comparisons | Avg. Speed-Up | recall | | |
|---|---|---|---|---|---|
| | | | @1 | @10 | @100 |
| W = 8 | 4,115 | **x 146** | 0.305 | 0.622 | 0.707 |
| W = 16 | 12,788 | x 65 | 0.343 | 0.742 | 0.886 |
| W = 32 | 37,951 | x 25 | 0.351 | 0.786 | 0.964 |
| W = 64 | 108,064 | x 9 | **0.352** | **0.795** | 0.986 |
| *Exhaustive* | 1,000,000 | x 1 | **0.352** | **0.795** | **0.988** |

As it can be seen, non-exhaustive search significantly decreases the number of comparisons, with a negligible drop in the performance. For example, for $W = 32$, almost exhaustive search performance is achieved for 25 times faster search. The overall search time is proportional to the number of comparisons, as long as the overhead of indexing is negligible compared to the overall search time. As the number of comparisons decreases, the search time also decreases. Hence the overhead is no longer negligible and the obtained speed-up is less compared to the decrease in the number of comparisons.

We also compare our method with the state-of-the-art non-exhaustive search methods in the literature. This time, the performance of our method is tested on SIFT1B dataset [10], which consists of 1 Billion samples. We compare against the state-of-the-art indexing based non-exhaustive search method *Locally Optimized Product Quantization* (*LOPQ*) [14], *Inverted File System with Asymmetric Distance Calculation* (*IVFADC*) [10] and IVFADC adaptation of OPQ (*I-OPQ*) [12], [14]. The compared methods use 64-bits for the codes and 13-bits for the coarse quantizer ($K = 8{,}192$), visiting the nearest 64 cells ($W = 64$). In order to use approximately the same amount of bits, we use 10 layers ($M = 10$) and the first two layers are used for indexing. The compared methods visit 64 cells among 8,192, which is approximately the same as visiting 22 cells for two layers of 256, hence we select $W = 22$ ($64/8{,}192 \cong 22^2/256^2$). The results are presented in Table 7. Here note that, the total

TABLE 7
recall@r Results for SIFT1B

| | #bits | recall@1 | recall@10 | recall@100 |
|---|---|---|---|---|
| *IVFADC* | 77 | 0.088 | 0.372 | 0.733 |
| *I-OPQ* | 77 | 0.114 | 0.399 | 0.777 |
| *LOPQ* | 77 | 0.199 | 0.586 | 0.909 |
| *CompQ* | 80 | **0.222** | **0.626** | **0.914** |

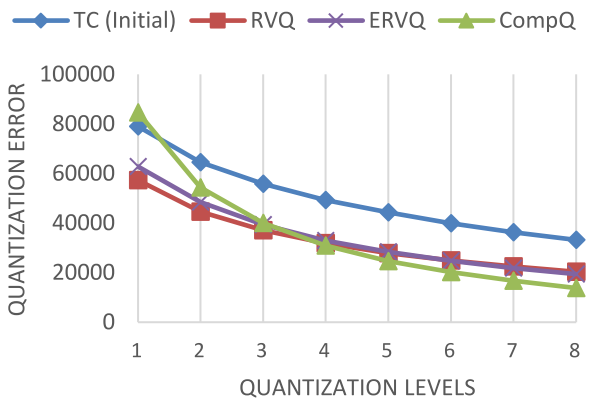


Fig. 2. Quantization error on SIFT1M as a function of the quantization levels for 64-bit encoding.

length of the binary code is not different in non-exhaustive search tests, unlike the most common cases in the literature, where the coarse quantizer indexes are calculated as an extra cost, i.e., [10], [12], [14], [16], [26], [27].

As it can be seen, our algorithm outperforms the state-of-the-art methods for all three scores, using 3 more bits per sample. This corresponds to an increase in storage around 3.9 percent only.

## 4 Discussions

As explained in detail in Section 2, *CompQ* presents a quantization scheme based on the addition of several codevectors. A joint optimization scheme is proposed on top of a hierarchical structure. This structure is similar to the hierarchy of RVQ and its variants, but *CompQ* outperforms these methods thanks to the jointly computed codebooks. Other state-of-the-art methods such as OCKM and AQ also propose a joint optimization for codebook generation, but the lack of codebook hierarchy leads to more complex search spaces for encoding. OTQ and CQ instead, limit the search space with stronger constraints, decreasing the search complexity but resulting in inferior codebooks. *CompQ* is a viable compromise, where the search complexity is reduced by adopting a hierarchical structure while the codebook generation is improved with the proposed joint optimization scheme.

### 4.1 Joint Optimization

Investigating the behavior of the quantization errors in each layer is a good way to visualize the advantage of joint optimization against training all codebooks separately. As it can be seen in Fig. 2, the first layers reduce the quantization error the most, while the contributions of the last layers are more limited. It is worth mentioning that, obtaining a perfectly equal distribution of quantization error cannot be expected. Because at each layer, the residuals from the previous layers are quantized, hence the norms are decreased. However, a better distribution of the workload between the layers is shown to be possible with the proposed joint optimization scheme.

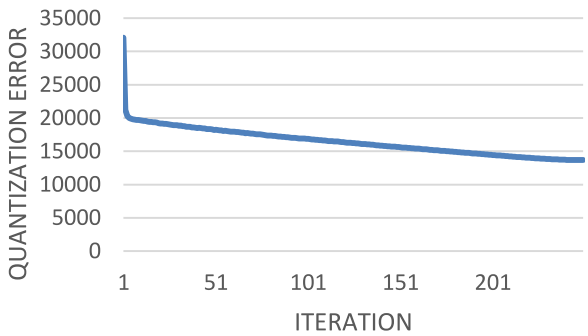


Fig. 3. Quantization error on SIFT1M as a function of training iterations for 64-bit encoding.

TABLE 8
Impact of the Proposed Novelties on the Performance
SIFT1M, 64-bit Codes

| | $MSE_Q$ | recall@1 | recall@10 | recall@100 |
|---|---|---|---|---|
| *RVQ* | 20302.1 | 0.257 | 0.659 | 0.952 |
| *CompQ* $H=1$ | 17765.9 | 0.286 | 0.709 | 0.966 |
| *RVQ* $H=8$ | 18735.3 | 0.298 | 0.726 | 0.973 |
| *CompQ* $H=8$ | 14418.1 | 0.339 | 0.783 | 0.983 |
| *CompQ* $H=16$ | 13964.2 | 0.347 | 0.786 | 0.986 |
| *CompQ* $H=32$ | **13671.2** | **0.352** | **0.795** | **0.988** |

Fig. 3 shows the decrease in the quantization error for 250 iterations. As it can be seen, there is a steep drop right after the first iteration, then the error keeps decreasing slowly and converges around 13,700. That also shows that the proposed initialization prevents overfitting and allows further iterations to improve the quantization performance.

### 4.2 Impact of the Proposed Novelties on the Performance

In Table 8, the impact of the proposed novelties on the performance are individually presented. The joint optimization scheme for codebook generation is compared to codebook generation scheme of RVQ. Also the winner codevector improvement is tested with RVQ and the performance is compared with *CompQ*. Both novelties are shown to provide significant improvement.

### 4.3 Parameter Selection

*CompQ* consists of several parameters, such as the number of quantization levels $M$, the number of codevectors per layer $K$, the learning rate $\gamma$, and the number of candidates for the winner codevector, $H$. For $M$ and $K$ we follow the same settings as in the literature, i.e., $M = 8$ for 64-bits code and $M = 4$ for 32-bits code. Similarly, we set $K = 256$.

Selecting a suitable number for the candidates of winner codevectors is a tradeoff between the encoding complexity and the quantization performance. The quantization performance increases with $H$ and so as the complexity. Hence we select $H = 32$ since it gives the comparable encoding complexity with the competeing methods as shown in Table 10. We present the performance of our method for different values of $H$ in Table 8. with codebooks trained with RVQ.

Note that when $H = 1$, the encoding scheme is the same as in RVQ. Table 8 also shows the increase in the performance when our encoding scheme is used. For the learning rates, since the upper layers correspond to greater quantization errors, the weight of the corresponding update should be also greater. In order to do that, we select the corresponding learning rate of each layer using the equation in (26).


Fig. 4. Quantization error on SIFT1M as a function of the learning rate parameter $\gamma_{Total}$ for 64-bit encoding, H = 1.

$$\gamma_m = \frac{1}{\log_2 m + 1}\gamma_0. \tag{26}$$

Then we define a total value $\gamma_{Total} = \sum \gamma_m$ and normalize $\gamma_m$ so that $\gamma_{Total}$ is equal to the predefined value. The quantization performance for different values of $\gamma_{Total}$ is presented in Fig. 4.

As it can be seen, there is no significant change in the quantization error for the tested values of $\gamma_{Total}$ and hence we pick $\gamma_{Total} = 0.5$ which corresponds to a slightly improved quantization error according to our simulations.

### 4.4 Initialization Methods

The importance of initialization for the proposed codebook training method is tested using different initialization schemes. Randomly initialized codebooks, RVQ initialized codebooks and the proposed codebook initialization, which is based on TC, are compared and the results are presented in Table 9.

TABLE 9
Comparison of Initialization Methods, SIFT1M, 64-bit Codes, H = 1

| | $MSE_Q$ | recall@1 | recall@10 | recall@100 |
|---|---|---|---|---|
| *CompQ Rand Init* | 21,568 | 0.227 | 0.614 | 0.931 |
| *CompQ RVQ Init* | 17,823 | 0.278 | 0.704 | 0.963 |
| *CompQ TC Init* | **17,765** | **0.286** | **0.709** | **0.966** |

As it can be seen, random initialization provides inferior results as some codevectors are not updated during the training procedure. RVQ initialization is also outperformed by the proposed TC initialization, as the former overfits the training data at each layer while the latter provides a more balanced distribution of codevectors throughout the feature space.

### 4.5 Computational Complexity Analysis

The computational cost of encoding for *CompQ* can be calculated as in (22). The first term in (22) is the distance of the given query to the codevectors of the previous layer, so at this layer no extra calculations are required. The third and fourth terms are obtained using a look-up table. This requires $mKH$ look-ups and additions for the $m$th layer. The second term is the dot product of the given query with

TABLE 10
Comparison of Computational and Storage Costs

| Method | Cost of Encoding | Cost of Encoding for Different Datasets and Code Lengths | | | |
|---|---|---|---|---|---|
| | | SIFT1M-32 | SIFT1M-64 | GIST1M-32 | GIST1M-64 |
| RVQ/ERVQ | $O(MKD)$ | 131,072 | 262,144 | 983,040 | 1,966,080 |
| CKM/OPQ | $O(D^2 + KD)$ | **49,152** | **49,152** | 1,167,360 | 1,167,360 |
| OCK | $O(TKD)$ | 327,680 | 327,680 | 2,457,600 | 2,457,600 |
| AQ | $O(M^2K^2(M + \log(MK)) + KD)$ | 14,712,832 | 79,724,544 | 14,925,824 | 79,937,536 |
| APQ | $O(D^2 + \frac{M}{4}(4^2K^2(4 + \log(4K)) + KD))$ | 14,729,216 | 14,761,984 | 15,847,424 | 16,093,184 |
| CQ | $O(3MKD)$ | 393,216 | 786,432 | 2,949,120 | 5,898,240 |
| OTQ | $O(D^2 + KD + MK^2)$ | 311,296 | 573,440 | 1,429,504 | 1,691,648 |
| KSSQ | $O(KD + 2\mathcal{K}DL + 8\mathcal{K}M)$ | 115,200 | 197,632 | **338,176** | **645,632** |
| CompQ | $O(MDK + \frac{(M-1)(M-2)}{2}KH + MKH \log H)$ | 319,488 | 761,856 | 1,171,456 | 2,465,792 |

| Method | Storage Cost | Storage Costs for Different Datasets and Code Lengths (MB) | | | |
|---|---|---|---|---|---|
| | | SIFT1M-32 | SIFT1M-64 | GIST1M-32 | GIST1M-64 |
| RVQ/ERVQ | $O(MKD)$ | 1.00 | 2.00 | 7.5 | 15 |
| CKM/OPQ | $O(D^2 + KD)$ | **0.38** | **0.38** | 8.91 | **8.91** |
| OCK | $O(D^2 + 2KD)$ | 0.63 | 0.63 | 10.78 | 10.78 |
| AQ | $O(MKD)$ | 1.00 | 2.00 | 7.50 | 15.00 |
| APQ | $O(D^2 + MKD)$ | 1.13 | 2.13 | 14.53 | 22.03 |
| CQ | $O(MKD)$ | 1.00 | 2.00 | 7.50 | 15.00 |
| OTQ | $O(D^2 + MKD)$ | 1.13 | 2.13 | 14.53 | 22.03 |
| KSSQ | $O(KDL)$ | 5.00 | 10.00 | **4.69** | 9.38 |
| CompQ | $O(MKD)$ | 1.00 | 2.00 | 7.50 | 15.00 |
| $K$: number of sub-codewords | | 256 | 256 | 256 | 256 |
| $D$: number of dimensions | | 128 | 128 | 960 | 960 |
| $M$: number of sub-codebooks | | 4 | 8 | 4 | 8 |
| $T$: search depth for OCK | | 10 | 10 | 10 | 10 |
| $H$: number of winner candidates | | 32 | 32 | 32 | 32 |
| $\mathcal{K}$: number of selected subspaces for KSSQ encoding | | 16 | 16 | 8 | 8 |
| $L$: number of reduced dimensions (on average) for KSSQ | | 20 | 40 | 20 | 40 |

each codevector of the current layer, which costs $O(DK)$. Finally, among the distances calculated between all codevectors and the previous residual candidates, the best $H$ are selected. This operation costs $O(KH \log H)$. These calculations are repeated for all $M$ layers, and the code corresponding to the best quantization error is returned. The final cost can be expressed as,

$$O\left(MDK + \frac{(M-1)(M-2)}{2}KH + MKH \log H\right). \quad (27)$$

As it can be seen in Table 10, the computational cost of our method is comparable to the other methods, but significantly lower than AQ/APQ. A detailed analysis of the storage requirements is also presented in Table 10. *CompQ* requires to store $K$ vectors of dimension $D$ for each of the $M$ layers, resulting in a storage cost of $O(MDK)$, which corresponds to $2$ MB for the SIFT dataset for 64-bits encoding. It can be observed from the table that the storage requirement of our method is also comparable with the other methods.

### 4.6 Relations with the Enhanced Residual Vector Quantization Method

ERVQ [17] is an extension over RVQ [16], which aims to improve the quantization quality by an iterative enhancement process over RVQ's training scheme. ERVQ starts with an RVQ initialization, and after that while keeping $M-1$ codebooks fixed, it recalculates the codebook of the $M$th layer. Several iterations are performed until convergence is reached. Compared to ERVQ, the proposed method has three significant differences. The first one is the proposed initialization scheme, which is based on TC [11] instead of RVQ. Table 9 shows the improvement in the performance provided by the TC based initialization method. The second difference is the training scheme. While ERVQ can only update one codebook per iteration, in the proposed method, all $M$ codebooks are updated according to the quantization error, as explained in Section 2.1, using the formula given in (18). The contribution of the proposed training method on RVQ is shown in Table 8. As it can be seen, the proposed method already performs better than ERVQ with these two improvements. The third and the last difference between the proposed method and ERVQ is the encoding scheme. The contribution of the proposed encoding scheme is also shown in in Table 8. Combining these three new features, the proposed method outperforms ERVQ with an important margin (a relative improvement of 27.5 percent for SIFT1M and 34.8 percent for GIST1M), clearly demonstrating the significance of the proposed contributions.

## 5 Conclusion

In this paper, a novel vector quantization method based on the addition of codevectors is presented. A novel training algorithm, which minimizes the quantization error using a joint optimization among different codebook layers is proposed. The proposed method, *CompQ*, also improves the traditional encoding scheme of RVQ by redefining the winner codevector. By means of such novel improvements, *CompQ* achieves the state-of-the-art performance with comparable computational and storage costs. In the future, we aim to test the performance of *CompQ* for different distance metrics and for k-Nearest Neighbor classification problems.

## References


[1] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," arXiv preprint arXiv:1408.2927, 2014.

[2] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on P-stable distributions," in *Proc. 29th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.

[3] X. He, D. Cai, S. Yan, and H. Zhang, "Neighborhood preserving embedding," in *Proc. 10th IEEE Int. Conf. Comput. Vis.*, 2005, pp. 1208–1213.

[4] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Neural Inf. Process. Syst.*, 2009, pp. 1753–1760.

[5] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Lett.*, vol. 31, no. 11, pp. 1348–1358, Aug. 2010.

[6] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2011, pp. 817–824.

[7] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2010, pp. 18–25.

[8] G. Lin, C. Shen, D. Suter, and A. van den Hengel, "A general two-step approach to learning-based hashing," in *Proc. 14th Int. Conf. Comput. Vis.*, 2013, pp. 2552–2559.

[9] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.

[10] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, Jan. 2011.

[11] J. Brandt, "Transform coding for fast approximate nearest neighbor search in high dimensions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2010, pp. 1815–1822.

[12] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 744–755, Dec. 2014.

[13] M. Norouzi and D. J. Fleet, "Cartesian k-means," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2013, pp. 3017–3024.

[14] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2014, pp. 2329–2336.

[15] E. C. Ozan, S. Kiranyaz, and M. Gabbouj, "K-subspaces quantization for approximate nearest neighbor search," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1722–1733, Jul. 2016.

[16] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.

[17] L. Ai, J. Yu, Z. Wu, Y. He, and T. Guan, "Optimized residual vector quantization for efficient approximate nearest neighbor search," in *Proc. Multimedia Syst.*, Jun. 2015, pp. 1–13.

[18] B. Wei, T. Guan, and J. Yu, "Projected residual vector quantization for ANN search," *IEEE Multimedia*, vol. 21, no. 3, pp. 41–51, Jul. 2014.

[19] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2014, pp. 931–938.

[20] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li, "Optimized Cartesian k-means," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 180–192, Jan. 2015.

[21] T. Zhang, D. Chao, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 838–846.

[22] A. Babenko and V. Lempitsky, "Tree quantization for large-scale similarity search and classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2015, pp. 4240–4248.

[23] J. A. Hertz, A. S. Krogh, R. G. Palmer, and A. S. Weigend, *Introduction to the Theory of Neural Computation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co. 1991.

[24] W. Dong, M. Charikar, and K. Li, "Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces," in *Proc. 31st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2008, pp. 123–130.

[25] J.-P. Heo, Z. Lin, and S.-E. Yoon, "Distance encoded product quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2014, pp. 2139–2146.
[26] A. Babenko and V. Lempitsky, "The inverted multi-index," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, 2012, vol. 14, no. 1-3, pp. 3069–3076.
[27] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," in *Proc. Int. Conf. Acoust. Speech Signal Process.*, pp. 861–864, 2011.
[28] J. Song, H. T. Shen, J. Wang, Z. Huang, N. Sebe, and J. Wang, "A distance-computation-free search scheme for binary code databases," *IEEE Trans. Multimedia*, vol. 18, no. 3, pp. 484–495, Mar. 2016.
[29] M. Norouzi, A. Punjani and D. J. Fleet, "Fast search in hamming space with multi-index hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1107–1119, Jun. 2014.

**Ezgi Can Ozan** received the BS degree from the Electrical and Electronics Department, Middle East Technical University, Ankara, Turkey, in 2007 and the MS degree in signal processing from the same University, in 2011. He is currently working toward the PhD degree and working as a researcher with the Tampere University of Technology, Tampere, Finland. His areas of interest include large-scale multimedia search, pattern recognition, machine learning, and computer vision.

**Serkan Kiranyaz** received the BS degree from the Electrical and Electronics Department, Bilkent University, Ankara, Turkey, in 1994, the MS degree in signal and video processing from the same university, in 1996, and the PhD degree from the Tampere University of Technology; Institute of Signal Processing, in 2005 and Docency at 2007, respectively. He worked as a senior researcher in the Nokia Research Center and later in Nokia Mobile Phones, Tampere, Finland. He is currently working as a professor in the Electrical Engineering Department, Qatar University. He published two books, more than 35 journal papers in several IEEE Transactions and some other high impact journals and 80+ papers in international conferences. His recent publication, "Automatic Object Segmentation by Quantum Cuts" won the IBM Best Paper Award at ICPR'14. He is a senior member of the IEEE.

**Moncef Gabbouj** received the BS degree in electrical engineering in 1985 from Oklahoma State University, Stillwater, and the MS and PhD degrees in electrical engineering from Purdue University, West Lafayette, Indiana, in 1986 and 1989, respectively. He is a professor of signal processing in the Department of Signal Processing, Tampere University of Technology, Tampere, Finland. He was an Academy of Finland professor during 2011-2015. He held several visiting professorships at different universities. His research interests include multimedia content-based analysis, indexing and retrieval, machine learning, nonlinear signal and image processing and analysis, voice conversion, and video processing and coding. He is a fellow of the IEEE and member of the Academia Europa and the Finnish Academy of Science and Letters. He is the past chairman of the IEEE CAS TC on DSP and committee member of the IEEE Fourier Award for Signal Processing. He served as distinguished lecturer for the IEEE CASS. He served as associate editor and guest editor of many IEEE and international journals. He was the recipient of the 2015 TUT Foundation Grand Award, the 2012 Nokia Foundation Visiting Professor Award, the 2005 Nokia Foundation Recognition Award, and several Best Paper Awards. He published more than 650 publications and supervised 40 doctoral theses.