Journal of Computer Science and Cybernetics, V.35, N.1 (2019), 69–83
DOI 10.15625/1813-9663/35/1/13442

PRODUCT SUB-VECTOR QUANTIZATION FOR FEATURE INDEXING

THE-ANH PHAM[1,*], DINH-NGHIEP LE[1], THI-LAN-PHUONG NGUYEN[2]

[1] *Hong Duc University*
[2] *Thai Nguyen University Lao Cai Campus,*
[*] *phamtheanh@hdu.edu.vn*

Abstract. This work addresses the problem of feature indexing to significantly accelerate the matching process which is commonly known as a cumbersome task in many computer vision applications. To this aim, we propose to perform product sub-vector quantization (PSVQ) to create finer representation of underlying data while still maintaining reasonable memory allocation. In addition, the quantized data can be jointly used with a clustering tree to perform approximate nearest search very efficiently. Experimental results demonstrate the superiority of the proposed method for different datasets in comparison with other methods.

Keywords. Product quantization; Hierarchical clustering tree; Approximate nearest search.

1. INTRODUCTION

Feature indexing has been known as an important technique which allows fast retrieval and matching of visual objects in computer vision filed. The most active application of feature indexing is probably concerned with fast approximate nearest neighbor (ANN) search. In the literature, popular approaches for this problem can be listed as space partitioning methods (typically, KD-tree [5] and randomized KD-trees [26], or LM-tree [24]), hashing methods (such as LSH [8], Kernelized LSH [12]), hierarchical clustering methods (such as vocabulary K-means tree [19], POC-trees [22]).

Recently, product quantization (PQ) [9] has been actively studied for its applications in fast approximate nearest neighbor search (ANN) and feature indexing. Different variants of PQ technique have been presented to optimize the quantization stage such as optimized PQ [6, 20], locally optimized PQ [10], or distribution sensitive PQ (DSPQ) [13]. PQ can be also combined with hierarchical clustering idea to boost the search performance as presented in [23, 25]. Extensive experiments have been conducted in [23, 25], demonstrating outstanding results of the combined PQ and K-means trees when comparing to existing approaches.

In this work, we propose exploiting a different usage of the PQ idea. As for the PQ, the data space is first partitioned into disjoint sub-spaces. Unlike PQ, the sub-vectors of several consecutive sub-spaces are grouped before performing vector quantization. This novel idea helps exploiting better the correlations of underlying data across the sub-spaces. In this way, several sub-spaces will share a common quantizer whose the number of centroids is much higher than those using in PQ. Specifically, the number of centroids or codewords used in our method is proportional to how many the sub-spaces is grouped. Although the proposed

method uses a higher number of codewords for each quantizer, the total number of centroids is still the same as those in PQ method and hence consuming the same amount of bit budget.

In addition, the present work also presents a novel way for dividing the space into smaller disjoint sub-spaces. Specifically, we take the advantage of prior knowledge of underlying feature spaces (i.e., SIFT and GIST features in our case) to figure out an optimized space decomposition scheme. The new splitting trick helps improving the coding quality greatly although it can not be applied to a general dataset.

The rest of this paper has been structured as follows. Section 2 reviews the-state-of-the-art techniques for feature indexing with a particular focus on product quantization approach and hierarchical clustering. Next, Section 3 describes in detail the proposed approach composing of two main contributions: the introduction of product sub-vector quantization and prior knowledge based space partitioning. Section 4 dedicates to the evaluation of the proposed method in comparison with other techniques. Finally, Section 5 concludes the paper and discusses several room of improvements for future research.

2. RELATED WORK

As the present work focuses on product quantization techniques, we shall discuss in this section the state-of-the-art methods that concern clustering tree and product quantization.

Hierarchical clustering. In a nutshell, a hierarchical clustering tree is created by iteratively dividing the underlying dataset into smaller sub-sets or clusters by using a standard clustering algorithm, typically K-means [15], K-medoids [11], or DBSCAN [4]. Each cluster is then further divided into sub-clusters until the resulting sub-sets are sufficiently small. To represent this hierarchical clustering decomposition, a tree structure is employed in which the root corresponds to the original dataset and each internal node represents a sub-cluster at the corresponding decomposition level. Each node of the tree also contains come basic information such as the centroid of the data points assigned to this node. Specially, the leaf nodes of the tree contains a bit richer information including the centroids and the indexes of data points.

One of the first hierarchical clustering structures is introduced by the authors in [19], namely vocabulary K-means tree. Tree building is performed exactly as described previously in which the K-means algorithm is chosen to cluster the feature vectors into smaller groups. Once the tree is created, it can be used to perform approximate nearest neighbor search (ANN) given a query, as follows. The search starts from the root node and goes down the tree. At each internal node, it selects a child to further explore by choosing the one having smallest Euclidean distance to the query. When positioning at a leaf node, linear scan is carried out to find the best candidate. Backtracking is then invoked to further refine the best answer. Depending on the application, the search may terminate early by posing a constraint on the maximum number of nodes to be visited.

There have been different progresses on improving the vocabulary K-means tree. Noticeably, the authors in [16, 18] equip the vocabulary K-means tree with a priority search strategy in which the search always selects the node from the top of a priory queue. This queue contains the list of candidate nodes stored in the increasing order of the distances to the query. Extensive experiments have showed outstanding results of this tree structure. In [22], the authors proposed a variant of clustering tree so-called Pair-wisely Optimized Clus-

tering tree (POC-tree). Each POC-tree is created in the same manner as the vocabulary K-means tree but differs in that it maximizes the separation space of every pair of clusters at each decomposition step. Hence, the resulting decomposition corresponds to a compact representation of the entire data space. Furthermore, multiple POC-trees can be combined to further improve the search performance, especially for highly dimensional feature space.

Product quantization. Product quantization (PQ) [9] is a powerful approach for data encoding and representation. In its essence, PQ divides the data space into disjoint sub-spaces and quantizes them separately. Specifically, for each sub-space, a sub-quantizer is learned by employing a standard vector quantizer such as K-means. Once the sub-quantizers have been trained, they are used to map an input vector into short codes, each of which corresponds to the index of a sub-codeword of a specific sub-quantizer. The short codes are then concatenated to from the complete code of the feature vector. To support ANN problem, PQ has been combined with an inverted file structure to achieve non-exhaustive search [9]. The resulting combination has demonstrates quite interesting results.

Since the first introduction of PQ technique, there has been an increasing interest of community to improve the PQ idea in different ways. The authors in [2] proposed attaching a separate inverted file structure to each sub-quantizer, hence resulting in a new technique so-called inverted multi-index. However, it was shown in their work that using more than two sub-quantizers is probably not a good choice. Hence, most part of the experiments have been reported for the case of second-order inverted multi-index only (i.e., using two sub-quantizers). For highly dimensional data space, this method could be seriously impacted by the curse-of-dimensionality problem because it is less benefited from using more than two sub-quantizers.

In the other side, the works in [6, 10, 20], have been presented with special focus on improving the coding quality of the quantization process. The main objective of those works here is to train the quantizers so as to adaptively capture the intrinsic distribution of the underlying data. To this aim, the work in [20] introduces two quantization techniques, namely ck-means and ok-means, which optimize the training step by making the data being rotated. As a result, the optimized model helps reducing significantly the quantization errors. Similarly, another work, so-called optimized product quantization (OPQ) [6], has been presented at the same time and shares the common idea. Here, OPQ first allows the data to be aligned by using PCA and then re-orders the dimensions to jointly achieve the two objectives of independence and balance between the sub-spaces. Those two features are so important when training the quantizers and they have been assumed to hold true in the PQ technique. OPQ works very well for single model distribution datasets but is less effective for the case of multi-model feature spaces. In the later case, locally OPQ (LOPQ) [10] is a reasonable choice because it applies OPQ locally on each cluster of data points. Nonetheless, the optimization process is done without taking into consideration the coarse vector quantization step.

Recently, the authors in [25] proposed combining the strength of PQ and hierarchical clustering resulting in a new technique, namely Embedding Product Quantization (EPQ). Similar to PQ, EQP divides the data space into disjoint sub-spaces. However, for each sub-space, a distinct vocabulary K-means tree is created to capture the distribution of underlying data. The tree can act as a sub-quantizer and an inverted file structure as well. Search performance has been proved to be very impressive in comparison with other techniques.

Another variant of EPQ has been introduced later in [23] by the same authors to further improve the search speedups.

Full-length quantization techniques. Interestingly, some recent works presented in [1, 3, 28, 29] make no assumption about the mutual independence as posed in PQ-based techniques. In [1], the authors present a new technique, namely Additive Quantization (AQ), which encodes each input vector as the sum of m codewords coming from m codebooks. Unlike PQ, the codewords are full-length as the original vectors and thus space decomposition is not done here. Another full-length encoding method has been presented in [3] by the same authors which encodes each vector by the means of a tree quantization (TQ). Each node of the tree corresponds to a codebook, while an edge is linked to a dimension. As a result, each codebook manages the quantization for a few dimensions that have been assigned to it only. The remaining dimensions are set to zero, resulting a quite sparse quantization vector. An optimized version of this technique (OTQ) has been also presented, where the data is globally rotated so as to achieve better fitting model. Experiments reported that those two techniques are superior to PQ in the means of coding quality but are less time efficient than the IVFADC [9].

To reduce the computational overhead of the AQ, TQ, and OTQ techniques, the work in [28] imposes an additional constraint for the quantization model. Particularly, it assumes that the product of inter-dictionary elements is a constant value. This extra assumption makes it simple the process of computing the distances between a query and the codewords. Furthermore, the authors also introduce a variant of this technique which exploits the sparsity of the codewords when training the codebooks [29]. The obtained model has been showed to be very effective when comparing to other methods.

3. THE PROPOSED APPROACH

3.1. Vector quantization and product sub-vector quantization

In the literature [6, 7, 9], vector quantization (VQ) refers to a pipeline process of building a codebook $\mathcal{C}$ composed of K codewords $\{c_1, c_2, \ldots, c_K\}$, and mapping a given input data point $x \in R^D$ to the nearest codeword. Formally, the map is denoted $q(x)$ as follows

$$q(x) \leftarrow \arg\min_{c_k \in \mathcal{C}} \mathbf{d}(x, c_k), \tag{1}$$

where $\mathbf{d}(x, c_k)$ denotes the Euclidean distance and $q(x)$ is so-called the quantizer.

To justify the quality of a quantizer, the quantization error is commonly used to measure the distortion of quantizing a dataset X composed of n data points in R^D space

$$E = \frac{1}{n} \sum_{x \in X} ||x - q(x)||^2, \tag{2}$$

where $||q - p||$ denotes the Euclidean norm between two points p and q. By conventionally, people [9] often employ the notation $||.||^2$ to compute the quantization or construction errors. We have thus adopted this notation in our manuscript.

The codebook $\mathcal{C}$ can be obtained by applying a standard clustering algorithm (e.g., K means) which partitions a set of data points in the R^D space into K clusters, each of which is represented by a centroid or codeword. By vector quantization, an input vector can be

represented by a very short code with the bit budget close to $\log(K)$. Hence, VQ has been extensively used to deal with many computer vision applications such as Bag-of-visual-words [27], fast nearest neighbor search [18].

Product quantization (PQ) [9], on the other hand, extends the idea of VQ in that the original data space is first divided into m distinct sub-spaces, followed by the VQ technique which is applied separately to the sub-vectors of each sub-space. Let $a_j(x)$ be the j^{th} sub-vector of an input vector $x \in R^D$ and $\mathcal{C}_j$ be the sub-codebook which is constructed from the j^{th} sub-space (here $j = 1, 2, \ldots, m$), a sub-quantizer $q_j(y)$ maps an input vector $y \in R^{D/m}$ to the nearest sub-codeword of $\mathcal{C}_j$ by

$$q_j(y) \leftarrow \arg\min_{c_{j,k} \in \mathcal{C}_j} \mathbf{d}(x, c_{j,k}), \tag{3}$$

where $c_{j,k}$ is the k^{th} sub-codeword of C_j.

In the PQ approach, each sub-codebook is commonly composed of K sub-codewords. The higher value of K, the finner the decomposition of underlying data. As a result, the quantization distortion is lower at the cost of increasing bit budget. Furthermore, as VQ is applied separately to each sub-space, the correlation of data across the sub-spaces is not exploited, leading to the redundant representation of sub-centroids. For instance, let us consider 512D GIST features, many sub-vectors (i.e., square blocks of the descriptor in Figure 1) in two consecutive sub-spaces are quite similar to each other. Those sub-vectors correspond to visual description of adjacent background blocks of an image. By building the sub-codebooks separately, those sub-vectors belong to the sub-centroids of different sub-spaces. Hence, many similar sub-centroids are created.

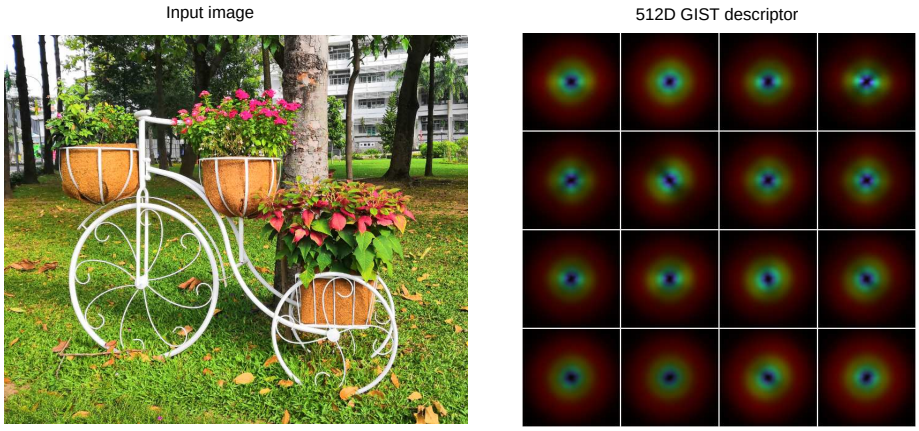


Figure 1. An input image (left) and its 512D GIST descriptor (right). Many background parts in image are similar in visual content leading to the similarity of descriptor blocks

To address these issues, we propose to gather the sub-vectors of continuous sub-spaces into bigger groups before applying vector quantization. Specifically, the proposed method, so-called product sub-vector quantization (PSVQ), combines the data from h, with $1 < h \leq m$, continuous sub-spaces, and performs vector quantization to create $m^* = m/h$ sub-quantizers. Each one consists $K^* = h \times K$ sub-centroids. Doing so, several sub-spaces shall share the same sub-quantizer and hence creating a finner decomposition of the underlying data. It is

worth noting that PSVQ does not increase the overall number of sub-codewords (i.e., there are still $m^* \times K^* = m \times K$ sub-codewords in total). Quantizing a sub-vector $y = a_j(x)$ belonging to the j^{th} sub-space ($1 \leq j \leq m$) is now processed by $q_i^*(y)$, with $i = \lfloor j/h \rfloor + 1$ and hence $1 \leq i \leq m^*$, as follows

$$q_i^*(y) \leftarrow \arg\min_{c_{i,k} \in \mathcal{C}_i^*} \mathbf{d}(y, c_{i,k}), \tag{4}$$

where $1 \leq k \leq K^*$, and $\mathcal{C}_i^*$ is the sub-codebook which is trained on a dataset composed of $n \times h$ data points, obtained by grouping the sub-vectors from the $(s+1)^{\text{th}}$ sub-space to the $(s+h)^{\text{th}}$ sub-space with $s = (i-1) \times h$.

All the process described above can be formulated on Algorithm 1. The algorithm takes as input a dataset X and several parameters, and then performs training process to create m^* sub-codebooks $\{\mathcal{C}_i^*\}$, each of which contains K^* sub-codewords. In its essence, the main computation aspect of Algorithm 1 is relied on the standard K-means algorithm. Its time and memory complexities are thus somewhat equivalent to those of K-means. It is worth noting that the main loop in Algorithm 1 is bounded by $m^* = m/h$ which is a small value ($m = 8$ in all of our experiments). In addition, as the training process is done in an offline phase, the time complexity is therefore not a major concern.

Given the sub-codebooks which are trained as aforementioned, the optimal quantization error (E^*) is now computed as follows

$$E^* = \frac{1}{n} \sum_{x \in X} \sum_{j=1}^{m} ||a_j(x) - q_i^*(a_j(x))||^2, \tag{5}$$

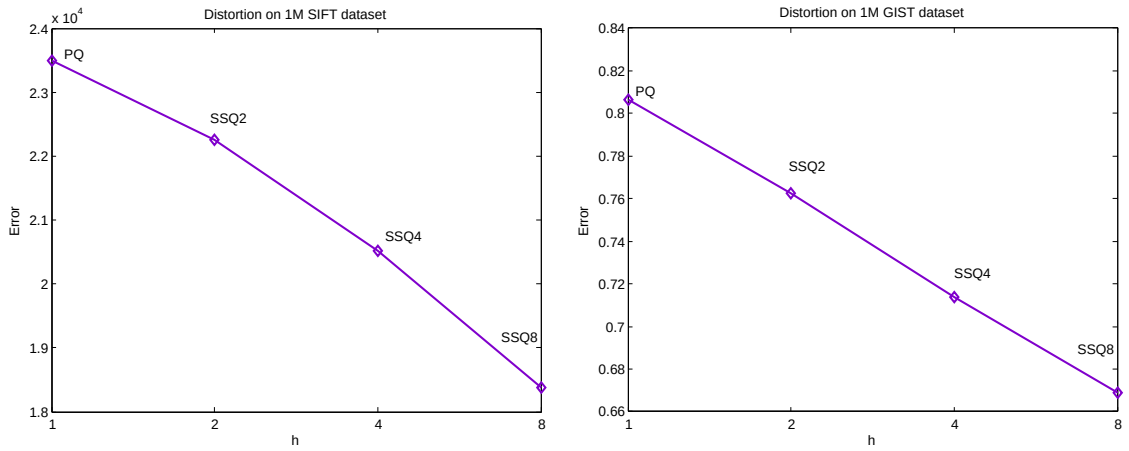
where $i = \lfloor j/h \rfloor + 1$ as described previously.

Algorithm 1 TrainPSVQ(m, X, h, K^*)

Input: m the number of sub-codebooks, X the dataset for training PSVQ sub-codebooks, h the number of sub-spaces to be merged, and K^* the number of sub-centroids in each sub-codebook.
Output: The list of m/h sub-codebooks.
$m^* \leftarrow m/h$
$L \leftarrow \emptyset$
Split X into m sub-sets: $X_1, X_2, \ldots, X_m$
for $i := 1$ **to** m^* **do**
    $s \leftarrow (i-1) \times h$
    $Y \leftarrow$ merge h continuous sub-sets $\{X_{s+1}, X_{s+2}, \ldots, X_{s+h}\}$
    $C_i^* \leftarrow$ Kmeans(Y, K^*) {apply K-means on the sub-set Y to create K^* sub-centers. Here, C_i^* contains K^* resulting sub-centers}
    $L \leftarrow$ Append(L, C_i^*) {append C_i^* into L}
end for
return L {L is an array of m^* sub-codebooks, each of which contains K^* sub-centers}

With this novel space decomposition, the data is presented at finer level because the similar sub-vectors in different sub-spaces are likely represented by a common centroid. In

doing so, we exploit more the correlation of data over the sub-spaces and make better fitting to the underlying data which finally results in lower coding distortion. Specifically, we shall show that the quantization error (E^*) is inversely proportional to the value of h. To this aim, we have trained the sub-quantizers on the training datasets (for SIFT and GIST features [9]) and then computed the errors for the database sets ($n = 1000000$ data points). Three different values of h (i.e., the number of sub-spaces for quantization or SSQ for short) are considered in this experiment, including $h = 2$ (SSQ2), $h = 4$ (SSQ4), $h = 8$ (SSQ8) and finally we set $m = 8$ as a common choice in the literature [9, 20]. Obviously, the case $h = 1$ corresponds to the standard PQ technique. As can be seen in Figure 2, the quantization error is greatly reduced when we go from PQ (i.e., $h = 1$) to SSQ2, SSQ4, and SSQ8 for both SIFT and GIST datasets.



(a) (b)

Figure 2. Quantization error for 1M SIFT (a) and 1M GIST (b) datasets

After training the sub-codebooks, they can be used to quantize a given feature database. Let G be a database composing of n points in R^d space and G_q be the quantization codes of G. Each quantized code of G_q is obtained by quantizing a data point $x \in \mathrm{G}$ based on the sub-codebooks $\{C_i^*\}$. More specifically, we first split x into m sub-vectors $a_j(x)$ with $1 \leq j \leq m$, and then find a sub-codeword of C_i^* $(i = \lfloor j/h \rfloor + 1)$ which is closest to $a_j(x)$ in terms of the Euclidean distance. The index associated to the obtained sub-codeword is considered as the sub-quantization of $a_j(x)$. As a result, the complete quantization code of x is an integer vector of size m whose each element has the value in the range of $0, 1, \ldots, K^*-1$. Repeating the above process for every data point in Q, we obtain the quantized database G_q which has the size of $n \times m$ and consumes much less memory space than the original float database G. Searching now can be accomplished by choosing an appropriate distance. For this purpose, asymmetric distance computation (ADC) is commonly used in the literature [9, 20]. By ADC distance, it means that we approximate the distance between two points x and r in the R^d space by the distance between x and $q^*(r)$ where $q^*(r)$ is a quantization mapping that concatenates the sub-quantization codes as follows

$$q^*(r) \leftarrow \{q_i^*(a_j(r))\}, \tag{6}$$

where $j = 1, 2, \ldots, m$ and $i = \lfloor j/h \rfloor + 1$.

With the use of ADC distance, the search procedure is outlined on Algorithm 2. The algorithm takes as input a query vector $x \in R^d$, a parameter R indicating the number of candidate answers, and the quantized database G_q. Principally, searching is driven by a exhaustive search which scans every element of G_q, computes the corresponding ADC distance, and updates a short-list containing R best answers found so far. The process of maintaining the short-list can be efficiently implemented by using a Maxheap structure [9] but it is not detailed here for simplification.

The main goal of Algorithm 2 is to justify the coding quality of our PSVQ quantization method. To this aim, running time aspect is not optimally addressed here and the algorithm acts as a brute-force search. The computation complexity is thus linear with the size of the database G_q. It is worth noting that the most intensively computational step of Algorithm 2 concerns with the ADC distance computation. However, the computational complexity of this step is bounded by the number of sub-codewords (i.e., $m \times K^*$) which is a relatively small value. We thus can pre-compute the distances between x and all the sub-codewords, store them into a 1D look-up table, and access the table for efficient calculation of ADC distance. The results of Algorithm 2 are compared against other coding methods such as PQ and ck-means. Detailed performance shall be presented in our subsequent experiments.

Algorithm 2 ADCSearch(x, R, G_q)

1: **Input**: an input query (x), the number of returned candidate answers (R), and the quantized database (G_q).
2: **Output**: A short-list of R elements closest to x using ADC distance.
3: $L_R \leftarrow \emptyset$
4: $n \leftarrow$ length of G_q
5: **for** $i := 1$ **to** n **do**
6: $\quad d \leftarrow \mathbf{d}(x, c_i)$ {compute the ADC distance between x and a quantized code, here c_i denotes the sub-codewords associated to the i^{th} quantized code of G_q}
7: $\quad L_R \leftarrow \mathrm{MaxHeap}(R, d, i)$ {update the list containing R best answers found so far}
8: **end for**
9: return L_R

3.2. Prior knowledge space partitioning

When performing space decomposition in the PQ technique, it is assumed that the sub-spaces are mutually independent. This is, of course, not always true for any dataset except for the SIFT features [14]. In the SIFT descriptor, the feature vector is computed for each local region in which the region is divided into 4×4 sub-windows, each of which is a 8-bin histogram of orientations computed for the pixels assigned to it. The histograms are then packed to form 128D feature vectors. Hence, if we divide the vectors into m sub-spaces (i.e., $m = 8, 16$), the resulting sub-spaces are virtually independent due to the natural order of SIFT dimensions. Similarly, the GIST features have been also computed with the same principle as illustrated in Figure 1 [21]. Particularly, in corresponding to the GIST1M dataset[1], the GIST features are computed for a given input image, as follows: resizing image

[1] http://corpus-texmex.irisa.fr/

to a fixed size, applying a 4×4 grid to each of three color channels, computing a 16-bin Gabor-based orientation histogram for each of 16 grid cells, and finally concatenating the resulting histograms to form 960D GIST features (i.e., $3 \times 16 \times 20$). However, the 960D GIST1M dataset has been reorganized with a slight modification in the order of concatenating the histograms. Therefore, the natural order of dimensions in GIST1M dataset does not correctly reflect the way of building GIST features.

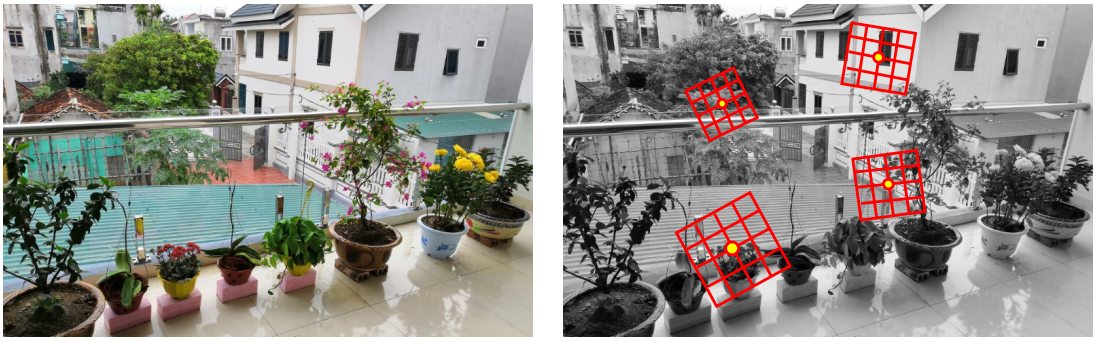


Figure 3. An input image (left) and SIFT descriptors computed at 4 keypoints (right)

Taking in consideration the aforementioned aspects, we argue that with some advance of prior knowledge about the feature space, we can employ a suitable space decomposition strategy. This observation has been also exploited in the PQ technique where the authors suggested using the "structured order" to split the datasets. By structured order, it means that all the dimensions having the same index modulo m ($m = 8$ in their work) are grouped into one sub-space. Experimental results showed that this idea gives dramatic improvement in coding quality for GIST features.

In the present work, we propose dividing the feature spaces in the same way as when we concatenate the histograms of SIFT and GIST features. As described before, for both the SIFT and GIST features, the input image region is divided into 4×4 sub-regions and then the statistical visual image content is separately collected for each sub-region. As a result, there is little correlation between the histograms of two sub-regions. In a nutshell, we divide the feature spaces into m sub-spaces ($m = 8$ in our experiments) where each sub-space contains the dimensions that have been assigned to the histograms computed from two continuous sub-regions for both SIFT and GIST features. Of course, if we set $m = 16$, the space decomposition would be very appropriable because each sub-space corresponds to a sub-region of the input image. However, we have chosen $m = 8$ to be comparable with the state-of-the-art PQ techniques.

4. EXPERIMENTS

4.1. Datasets and settings

To validate the performance of the proposed method, we have conducted two different experimental tests. The first test aims at comparing the coding quality or quantization performance of the proposed method with other PQ-based techniques that have particular

interest in optimizing the quantizers. To this aim, we have selected the two representative methods including the standard PQ and the ck-means method [20].

The second test compares the search efficiency or ANN search performance of the proposed method. Because the main goal here is to investigate the indexing aspect, we have thus selected the best indexing strategies including: IVFADC (i.e., PQ combined with an inverted file structure using the ADC distance [9]), K-means tree [16] (i.e., the most efficient technique for ANN search of FLANN library[2]), POC-tree [22], EPQ [25], and optimized EPQ (OEPQ) [23]. The source codes are in C/C++ environment and the test is carried out on a standard computer with following configuration: Windows 7, 16Gb RAM, Intel Core (Dual-Core) i7 2.1 GHz.

As for the evaluation metrics, the first test employs Recall@R protocol, while the second test uses speedup/precision curves. Those criteria are commonly used in the literature for the same tasks [9, 17, 18, 20]. Specifically, Recall@R measures the faction of corrected answers from a shortlist of R elements. The speedups are relatively computed over linear scan search for avoiding the impact of computer configuration. To have a stable report, we have run thousands of queries and then computed the mean results as the final output.

Two public datasets are used for all of our experiments. They include ANN_SIFT1M and ANN_GIST1M [9]. Some basic information is reported in Table 1 for both the datasets. For training the quantizers, PQ, ck-means, and the proposed method use the training set which is different from the database and test sets. In addition, all these methods also use the same parameter settings as follows: $m = 8$ and $K = 256$. Differently, the remaining methods (i.e., K-means tree, POC-tree, EPQ and OEPQ) use directly the database for training the quantizers (if any) and the clustering trees as well because they have been specifically targeted to the indexing aspect.

Table 1. The datasets used in our experiments

Dataset	#Training set	#Database	#Queries	#Dimension
ANN_SIFT1M	100,000	1,000,000	10,000	128
ANN_GIST1M	500,000	1,000,000	1000	960

4.2. Evaluation of coding quality

We compare the proposed method against PQ and ck-means in terms of coding quality for different sceneries of h as mentioned before, including SSQ2, SSQ4, and SSQ8 which correspond to 4, 2, and 1 sub-quantizers, respectively. For instances, using $h = 2$ (SSQ2) means that we group the data of two successive sub-spaces before applying the vector quantization process and thus resulting in 4 sub-quantizers in total. To this aim, Algorithm 2 is employed to compute the Recall@R scores of our algorithm. Figure 4 presents the results of all the methods for both SIFT and GIST datasets. One can observe that SSQ2, SSQ4, and SSQ8 perform consistently better than both PQ and ck-means for SIFT features. For GIST dataset, ck-means is only outperformed by our SSQ8 variant probably due to the impact of

[2]http://www.cs.ubc.ca/research/flann/

highly dimensional effect. These results are quite impressive when considering that ck-means has been fully optimized for capturing the intrinsic distribution of underlying data.

On the other hand, Figure 4 also shows that PQ when combined with the structured order strategy for space decomposition is still not superior to our simplest model SSQ2. Obviously, the SSQ8 variant performs best because the correlation of data is exploited at maximum over all the sub-spaces, resulting in the lowest coding distortion as empirically presented in Figure 2. However, this superior coding quality does not necessarily imply the improvement in search efficiencies because of the computational overhead when computing the distances. As a result, one has to make a trade-off between both these aspects depending on specific application context.

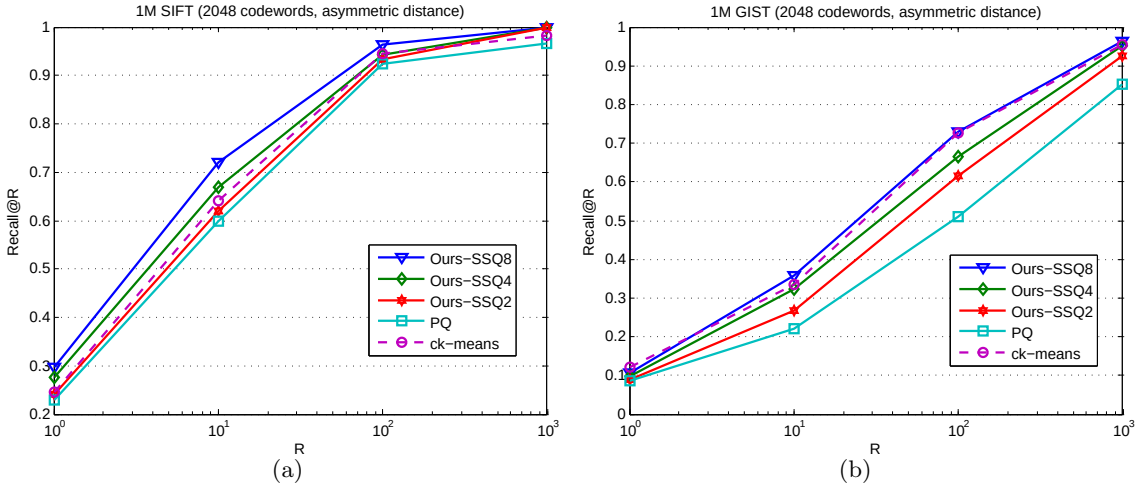


Figure 4. Coding quality for SIFT (a) and GIST (b) features

4.3. Evaluation of approximate nearest neighbor search

In this part, we investigate the efficiency of the proposed sub-quantizers for the task of ANN search. To this aim, we employ the indexing strategy as presented by the authors in [23]. This protocol is a standard pipeline composed of a clustering tree for hierarchical representation of the whole database, and a heuristic engine for driving the search for a given query. This indexing structure was coupled with the PQ-based quantizers and have employed ADC distances to return an ordered list of candidate answers. To the best of our understanding, this method currently yields the state-the-art results for ANN search on the studied datasets. However, these results have been obtained with a constraint that the data space for GIST features is divided into many small sub-spaces to avoid the problem of highly dimensional effect.

Differing from the aforementioned work, the main goal of this study is to evaluate the search efficiency when using even much simple settings (i.e., the data space is divided into 8 sub-spaces for both SIFT ad GIST features). As mentioned before, ANN search performance is measured by the means of speedup/precision curves. Here, the search precision can be considered as a special case of Recall@R where $R = 1$. That means the fraction of true

answers when considering the short-list containing only one candidate. To obtain a wide range of speedup/precision points, several parameters are varied when performing the query. More detail is referred to original work [23].

Figure 5 compares the ANN search efficiencies of our method against recent works including EPQ, OEPQ, POC-tree, and Kmeans-tree. As shown from Figure 5, the proposed method performs on par with the state-of-the-art indexing techniques (i.e., OEPQ and EPQ) for the SIFT features, and especially outperforms all other methods for GIST dataset. On average, the proposed method is about 2.5× to 3.0× faster than the best algorithm of FLANN library (i.e., Kmeans-tree). The gain in speedups is even higher for GIST features. Taking a particular snapshot at the search precision of 85%, our algorithm is 500× and 365× faster than sequence search for GIST and SIFT, respectively. Obviously, one can even obtain further interesting results by dividing the data space into more than 8 sub-spaces ($m = 16$ for example). However, we have not conducted this option in the present work.

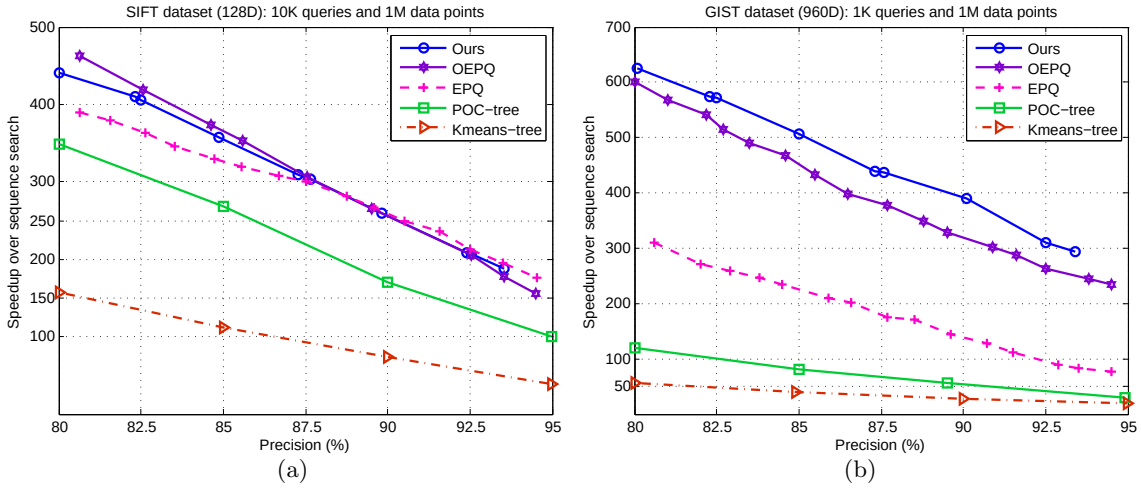


Figure 5. ANN search performance for SIFT (a) and GIST (b) features

Lastly, we also provide a short evaluation of our method with the IVFADC technique [9] by comparing the search speedups relatively to the best algorithm of FLANN as suggested in [23]. One advantage of this way is that one can avoid the barrier and subjective evaluation caused by re-implementing the algorithm and parameter tuning. As reported by the authors in [9] and [23], the speedup of IVFADC is about 1.5× to 2.0× higher than the best algorithm of FLANN for the same precisions on SIFT dataset, while the proposed method, as noticed before, is roughly 2.5× to 3.0× more efficient. The obtained results are quite impressive which demonstrates the efficiency and effectiveness of the proposed quantizers although the idea is conceptually simple.

5. CONCLUSIONS

The proposed method has been designed to exploit the correlation of underlying data by allowing a centroid to be associated to more than one sub-space. Hence, the density of each cluster is higher when comparing to individual quantizer created separately in each sub-space.

In addition, the proposed method also takes into consideration the prior knowledge of data distribution to decide the appropriate space splitting action. The resulting quantizers have nice properties in that they yield lower coding errors, create finer data representation, and consume same bit budget for storing the sub-codewords as standard PQ methods do. Various experiments have been conducted which showed the superior performance of the proposed method in comparison with other techniques. For extension of this work, we intend to study the incorporation of deep learning approach for boosting the search performance.

ACKNOWLEDGMENT

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2016.01.

REFERENCES

[1] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 931–938.

[2] ——, "The inverted multi-index," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 6, pp. 1247–1260, 2015.

[3] ——, "Tree quantization for large-scale similarity search and classification," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4240–4248.

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231.

[5] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[6] 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, 2014.

[7] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[8] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC'98, 1998, pp. 604–613.

[9] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, 2011.

[10] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR 2014)*, Columbus, Ohio, June 2014, pp. 2329–2336.

[11] L. Kaufman and P. J. Rousseeuw, "Clustering by means of medoids," pp. 405–416, 1987.

[12] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *IEEE International Conference on Computer Vision*, ser. ICCV'09, 2009, pp. 2130–2137.

[13] L. Li, Q. Hu, Y. Han, and X. Li, "Distribution sensitive product quantization," *IEEE Transactions on Circuits and Systems for Video Technology. https://doi.org/10.1109/TCSVT.2017.2759277*, 2017.

[14] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[15] J. B. Macqueen, "Some methods for classification and analysis of multivariate observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[16] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proceedings of International Conference on Computer Vision Theory and Applications*, ser. VISAPP'09, 2009, pp. 331–340.

[17] ——, "Fast matching of binary features," in *Proceedings of the Ninth Conference on Computer and Robot Vision*, ser. CRV'12, 2012, pp. 404–410.

[18] ——, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, pp. 2227–2240, 2014.

[19] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, ser. CVPR'06, 2006, pp. 2161–2168.

[20] M. Norouzi and D. J. Fleet, "Cartesian k-means," in *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '13, 2013, pp. 3017–3024.

[21] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[22] T.-A. Pham, "Pair-wisely optimized clustering tree for feature indexing," *Computer Vision and Image Understanding*, vol. 154, no. 1, pp. 35–47, 2017.

[23] ——, "Improved embedding product quantization," *Machine Vision and Applications, Published online 13 Feb 2019. https://doi.org/10.1007/s00138-018-00999-2*, 2018.

[24] T.-A. Pham, S. Barrat, M. Delalandre, and J.-Y. Ramel, "An efficient indexing scheme based on linked-node m-ary tree structure," in *17th International Conference on Image Analysis and Processing (ICIAP 2013)*, ser. LNCS, vol. 8156, 2013, pp. 752–762.

[25] T.-A. Pham and N.-T. Do, "Embedding hierarchical clustering in product quantization for feature indexing," *Multimedia Tools and Applications. https://doi.org/10.1007/s11042-018-6626-9*, 2018.

[26] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR'08, 2008, pp. 1–8.

[27] J. Willamowski, D. Arregui, G. Csurka, C. R. Dance, and L. Fan, "Categorizing nine visual classes using local appearance descriptors," in *In ICPR Workshop on Learning for Adaptable Visual Systems*, 2004.

[28] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 838–846.

[29] T. Zhang, G.-J. Qi, J. Tang, and J. Wang, "Sparse composite quantization," in *Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR'15)*, 2015, pp. 4548–4556.

Received on December 14, 2018
Revised on March 10, 2019