Locally Optimized Product Quantization for Approximate Nearest Neighbor Search

Yannis Kalantidis and Yannis Avrithis
National Technical University of Athens
{ykalant, iavr}@image.ntua.gr

Abstract

We present a simple vector quantizer that combines low distortion with fast search and apply it to approximate nearest neighbor (ANN) search in high dimensional spaces. Leveraging the very same data structure that is used to provide non-exhaustive search, i.e.*, inverted lists or a multi-index, the idea is to* locally *optimize an individual product quantizer (PQ) per cell and use it to encode residuals. Local optimization is over rotation and space decomposition; interestingly, we apply a parametric solution that assumes a normal distribution and is extremely fast to train. With a reasonable space and time overhead that is constant in the data size, we set a new state-of-the-art on several public datasets, including a billion-scale one.*

1. Introduction

Approximate nearest neighbor (ANN) search in high-dimensional spaces is not only a recurring problem in computer vision, but also undergoing significant progress. A large body of methods maintain all data points in memory and rely on efficient data structures to compute only a limited number of exact distances, that is ideally fixed [14]. At the other extreme, mapping data points to compact binary codes is not only efficient in space but may achieve fast exhaustive search in Hamming space [10, 16].

Product quantization (PQ) [12] is an alternative compact encoding method that is discrete but not binary and can be used for exhaustive or non-exhaustive search through inverted indexing or multi-indexing [3]. As is true for most hashing methods [11], better fitting to the underlying distribution is critical in search performance, and one such approach for PQ is *optimized product quantization* (OPQ) [9] and its equivalent *Cartesian k-means* [15].

How are such training methods beneficial? Different criteria are applicable, but the underlying principle is that *all bits allocated to data points should be used sparingly*. Since search can be made fast, such methods should be ultimately seen as (lossy) *data compression* targeting minimal distortion, with extreme examples being [1, 5].

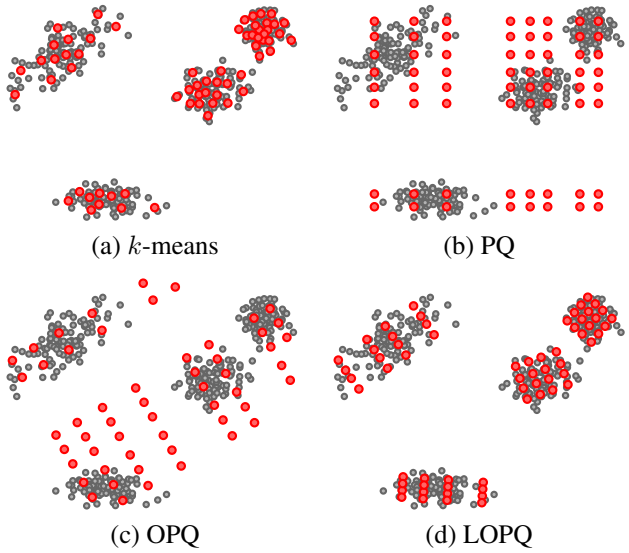

Figure 1. Four quantizers of 64 centroids (•) each, trained on a random set of 2D points (•), following a mixture distribution. (c) and (d) also reorder dimensions, which is not shown in 2D.

As such, k-means, depicted in Fig. 1(a), is a vector quantization method where by specifying k centroids, $\log_2 k$ bits can represent an arbitrary data point in $\mathbb{R}^d$ for any dimension d; but naïve search is $O(dk)$ and low distortion means very large k. By constraining centroids on an axis-aligned, m-dimensional grid, PQ achieves k^m centroids keeping search at $O(dk)$; but as illustrated in Fig. 1(b), many of these centroids remain without data support *e.g.* if the distributions on m subspaces are not independent.

OPQ allows the grid to undergo arbitrary rotation and reordering of dimensions to better align to data and balance their variance across subspaces to match bit allocation that is also balanced. But as shown in Fig. 1(c), a strongly multimodal distribution may not benefit from such alignment.

Our solution in this work is *locally optimized product*

quantization (LOPQ). Following a quite common search option of [12], a coarse quantizer is used to index data by inverted lists, and residuals between data points and centroids are PQ-encoded. But within-cell distributions are largely unimodal; hence, as in Fig. 1(d), we locally optimize an individual product quantizer per cell. Under no assumptions on the distribution, practically all centroids are supported by data, contributing to a lower distortion.

LOPQ requires reasonable space and time overhead compared to PQ, both for offline training/indexing, and online queries; but all overhead is constant in data size. It is embarrassingly simple to apply and boosts performance on several public datasets. A multi-index is essential for large scale datasets and combining with LOPQ is less trivial, but we provide a scalable solution nevertheless.

2. Related work and contribution

Focusing on large datasets where index space is the bottleneck, we exclude *e.g.* tree-based methods like [14] that require all data uncompressed in memory. Binary encoding is the most compact representation, approaching ANN search via exhaustive search in Hamming space. Methods like spectral hashing [18], ITQ [10] or k-means hashing [11] focus on learning optimized codes on the underlying data distribution. Search in Hamming space is really fast but, despite learning, performance suffers.

Significant benefit is to be gained via multiple quantizers or hash tables as in LSH [6], at the cost of storing each point index multiple times. For instance, [17, 19] gain performance by multiple k-means quantizers via random reinitializing or partitioning jointly trained centroids. Similarly, multi-index hashing [16] gains speed via multiple hash tables on binary code substrings. We still outperform all such approaches *at only a fraction of index space*.

PQ [12] provides efficient vector quantization with less distortion than binary encoding. Transform coding [4] is a special case of scalar quantization that additionally allocates bits according to variance per dimension. OPQ [9] and Ck-means [15] generalize PQ by jointly optimizing rotation, subspace decomposition and sub-quantizers. Interestingly, the parametric solution of OPQ aims at the exact opposite of [4]: balancing variance given a uniform bit allocation over subspaces.

Although [12] provides non-exhaustive variant IVFADC based on a coarse quantizer and PQ-encoded residuals, [9, 15] are exhaustive. The inverted *multi-index* [3] achieves very fine space partitioning via one quantizer per subspace and is compatible with PQ-encoding, gaining performance at query times comparable to Hamming space search. On the other hand, the idea of space decomposition can be applied recursively to provide extremely fast codebook training and vector quantization [2].

The recent extension of OPQ [8] combines optimization with a multi-index and is the current state-of-the-art on a billion-scale dataset, but all optimizations are still global. We observe that OPQ performs significantly better when the underlying distribution is unimodal, while residuals are much more unimodal than original data. Hence we independently optimize per cell to distribute centroids mostly over underlying data, despite the constraints of a product quantizer. In particular, we make the following contributions:

1. Partitioning data in cells, we *locally optimize* one product quantizer per cell on the residual distribution.
2. We show that training is practical since local distributions are easier to optimize via a simple OPQ variant.
3. We provide solutions for either a *single* or a *multi-index*, fitting naturally to existing search frameworks for state-of-the-art performance with little overhead.

A recent related work is [7], applying a local PCA rotation per centroid prior to VLAD aggregation. However, both our experiments and [9, 8] show that PCA without subspace allocation actually damages ANN performance.

3. Background

Vector quantization. A *quantizer* is a function q that maps a d-dimensional vector $\mathbf{x} \in \mathbb{R}^d$ to vector $q(\mathbf{x}) \in \mathcal{C}$, where $\mathcal{C}$ is a finite subset of $\mathbb{R}^d$, of cardinality k. Each vector $\mathbf{c} \in \mathcal{C}$ is called a *centroid*, and $\mathcal{C}$ a *codebook*. Given a finite set $\mathcal{X}$ of data points in $\mathbb{R}^d$, q induces *distortion*

$$E = \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - q(\mathbf{x})\|^2. \tag{1}$$

According to Lloyd's first condition, regardless of the chosen codebook, a quantizer that minimizes distortion should map vector $\mathbf{x}$ to its nearest centroid, or

$$\mathbf{x} \mapsto q(\mathbf{x}) = \arg\min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|, \tag{2}$$

for $\mathbf{x} \in \mathbb{R}^d$. Hence, an optimal quantizer should minimize distortion E as a function of codebook $\mathcal{C}$ alone.

Product quantization. Assuming that dimension d is a multiple of m, write any vector $\mathbf{x} \in \mathbb{R}^d$ as a concatenation $(\mathbf{x}^1, \dots, \mathbf{x}^m)$ of m sub-vectors, each of dimension d/m. If $\mathcal{C}^1, \dots, \mathcal{C}^m$ are m sub-codebooks in subspace $\mathbb{R}^{d/m}$, each of k sub-centroids, a *product quantizer* [12] constrains $\mathcal{C}$ to the Cartesian product

$$\mathcal{C} = \mathcal{C}^1 \times \cdots \times \mathcal{C}^m, \tag{3}$$

i.e., a codebook of k^m centroids of the form $\mathbf{c} = (\mathbf{c}^1, \dots, \mathbf{c}^m)$ with each sub-centroid $\mathbf{c}^j \in \mathcal{C}^j$ for $j \in \mathcal{M} = \{1, \dots, m\}$. An optimal product quantizer q should minimize distortion E (1) as a function of $\mathcal{C}$, subject to $\mathcal{C}$ being of the form (3) [9]. In this case, for each $\mathbf{x} \in \mathbb{R}^d$, the nearest centroid in $\mathcal{C}$ is

$$q(\mathbf{x}) = (q^1(\mathbf{x}^1), \dots, q^m(\mathbf{x}^m)), \tag{4}$$

where $q^j(\mathbf{x}^j)$ is the nearest sub-centroid of sub-vector $\mathbf{x}^j$ in $\mathcal{C}^j$, for $j \in \mathcal{M}$ [9]. Hence an optimal product quantizer q in d dimensions incurs m subproblems of finding m optimal sub-quantizers $q^j, j \in \mathcal{M}$, each in d/m dimensions. We write $q = (q^1, \ldots, q^m)$ in this case.

Optimized product quantization [9],[15] refers to optimizing the subspace decomposition apart from the centroids. Constraint (3) of the codebook is now relaxed to

$$\mathcal{C} = \{R\hat{\mathbf{c}} : \hat{\mathbf{c}} \in \mathcal{C}^1 \times \cdots \times \mathcal{C}^m, R^{\mathrm{T}}R = I\}, \qquad (5)$$

where orthogonal $d \times d$ matrix R allows for arbitrary rotation and permutation of vector components. Hence E should be minimized as a function of $\mathcal{C}$, subject to $\mathcal{C}$ being of the form (5). Optimization with respect to R and $\mathcal{C}^1, \ldots, \mathcal{C}^m$ can be either joint as in Ck-means [15] and in the non-parametric solution OPQ_{NP} of [9], or decoupled, as in the parametric solution OPQ_{P} of [9].

Exhaustive search. Given a product quantizer $q = (q^1, \ldots, q^m)$, assume that each data point $\mathbf{x} \in \mathcal{X}$ is represented by $q(\mathbf{x})$ and encoded as tuple $(i^1, \ldots, i^m)$ of m sub-centroid indices (4), each in index set $\mathcal{K} = \{1, \ldots, k\}$. This PQ-*encoding* requires $m \log_2 k$ bits per point.

Given a new *query* vector $\mathbf{y}$, the (squared) Euclidean distance to every point $\mathbf{x} \in \mathcal{X}$ may be approximated by

$$\delta_q(\mathbf{y}, \mathbf{x}) = \|\mathbf{y} - q(\mathbf{x})\|^2 = \sum_{j=1}^{m} \|\mathbf{y}^j - q^j(\mathbf{x}^j)\|^2, \qquad (6)$$

where $q^j(\mathbf{x}^j) \in \mathcal{C}^j = \{\mathbf{c}_1^j, \ldots, \mathbf{c}_k^j\}$ for $j \in \mathcal{M}$. Distances $\|\mathbf{y}^j - \mathbf{c}_i^j\|^2$ are precomputed for $i \in \mathcal{K}$ and $j \in \mathcal{M}$, so (6) amounts to only $O(m)$ lookup and add operations. This is the asymmetric distance computation (ADC) of [12].

Indexing. When quantizing point $\mathbf{x} \in \mathbb{R}^d$ by quantizer q, its *residual* vector is defined as

$$r_q(\mathbf{x}) = \mathbf{x} - q(\mathbf{x}). \qquad (7)$$

Non-exhaustive search involves a *coarse quantizer* Q of K centroids, or *cells*. Each point $\mathbf{x} \in \mathcal{X}$ is quantized to $Q(\mathbf{x})$, and its residual vector $r_Q(\mathbf{x})$ is quantized by a product quantizer q. For each cell, an inverted list of data points is maintained, along with PQ-encoded residuals.

A query point $\mathbf{y}$ is first quantized to its w nearest cells, and approximate distances between residuals are then found according to (6) only within the corresponding w inverted lists. This is referred to as IVFADC search in [12].

Re-ranking. Second-order residuals may be employed along with ADC or IVFADC, again PQ-encoded by m' sub-quantizers. However, this requires full vector reconstruction, so is only used for re-ranking [13].

Multi-indexing applies the idea of PQ to the coarse quantizer used for indexing. A second-order inverted multi-index [3] comprises two *subspace quantizers* Q^1, Q^2 over $\mathbb{R}^{d/2}$, each of K sub-centroids. A *cell* is now a pair of sub-centroids. There are K^2 cells, which can be structured on a 2-dimensional *grid*, inducing a fine partition over $\mathbb{R}^d$. For each point $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2) \in \mathcal{X}$, sub-vectors $\mathbf{x}^1, \mathbf{x}^2 \in \mathbb{R}^{d/2}$ are separately (and exhaustively) quantized to $Q^1(\mathbf{x}^1), Q^2(\mathbf{x}^2)$, respectively. For each cell, an inverted list of data points is again maintained.

Given a query vector $\mathbf{y} = (\mathbf{y}^1, \mathbf{y}^2)$, the (squared) Euclidean distances of each of sub-vectors $\mathbf{y}^1, \mathbf{y}^2$ to all sub-centroids of Q^1, Q^2 respectively are found first. The distance of $\mathbf{y}$ to a cell may then be found by a lookup-add operation, similarly to (6) for $m = 2$. Cells are traversed in increasing order of distance to $\mathbf{y}$ by the *multi-sequence* algorithm [3]—a form of distance propagation on the grid—until a target number T of points is collected. Different options exist for encoding residuals and re-ranking.

4. Locally optimized product quantization

We investigate two solutions: ordinary inverted lists, and a second-order multi-index. Section 4.1 discusses LOPQ in the former case, which simply allocates data to cells and locally optimizes a product quantizer per cell to encode residuals. Optimization per cell is discussed in section 4.2, mostly following [9, 15]; the same process is used in section 4.4, discussing LOPQ in the multi-index case.

4.1. Searching on a single index

Given a set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of n data points in $\mathbb{R}^d$, we optimize a coarse quantizer Q, with associated codebook $\mathcal{E} = \{\mathbf{e}_1, \ldots, \mathbf{e}_K\}$ of K centroids, or cells. For $i \in \mathcal{K} = \{1, \ldots, K\}$, we construct an inverted list $\mathcal{L}_i$ containing indices of points quantized to cell $\mathbf{e}_i$,

$$\mathcal{L}_i = \{j \in \mathcal{N} : Q(\mathbf{x}_j) = \mathbf{e}_i\} \qquad (8)$$

where $\mathcal{N} = \{1, \ldots, n\}$, and collect their residuals in

$$\mathcal{Z}_i = \{\mathbf{x} - \mathbf{e}_i : \mathbf{x} \in \mathcal{X},\ Q(\mathbf{x}) = \mathbf{e}_i\}. \qquad (9)$$

For each cell $i \in \mathcal{K}$, we locally optimize PQ encoding of residuals in set $\mathcal{Z}_i$, as discussed in section 4.2, yielding an orthogonal matrix R_i and a product quantizer q_i. Residuals are then locally rotated by $\hat{\mathbf{z}} \leftarrow R_i^{\mathrm{T}}\mathbf{z}$ for $\mathbf{z} \in \mathcal{Z}_i$ and PQ-encoded as $q_i(\hat{\mathbf{z}}) = q_i(R_i^{\mathrm{T}}\mathbf{z})$.

At query time, the query point $\mathbf{y}$ is soft-assigned to its w nearest cells $\mathcal{A}$ in $\mathcal{E}$. For each cell $\mathbf{e}_i \in \mathcal{A}$, residual $\mathbf{y}_i = \mathbf{y} - \mathbf{e}_i$ is individually rotated by $\hat{\mathbf{y}}_i \leftarrow R_i^{\mathrm{T}}\mathbf{y}_i$. Asymmetric distances $\delta_{q_i}(\hat{\mathbf{y}}_i, \hat{\mathbf{z}}_p)$ to residuals $\hat{\mathbf{z}}_p$ for $p \in \mathcal{L}_i$ are then computed according to (6), using the underlying local product quantizer q_i. The computation is exhaustive within list $\mathcal{L}_i$, but is performed in the compressed domain.

Analysis. To illustrate the individual gain from the two optimized quantities, we investigate optimizing rotation alone

with fixed sub-quantizers, as well as both rotation and sub-quantizers, referred to as LOR+PQ and LOPQ, respectively. In the latter case, there is an $O(K(d^2+dk))$ space overhead, comparing *e.g.* to IVFADC [12]. Similarly, local rotation of the query residual imposes an $O(wd^2)$ time overhead.

4.2. Local optimization

Let $\mathcal{Z} \in \{\mathcal{Z}_1, \ldots, \mathcal{Z}_K\}$ be the set of residuals of data points quantized to some cell in $\mathcal{E}$. Contrary to [12], we PQ-encode these residuals by locally optimizing both space decomposition and sub-quantizers per cell. Given m and k as parameters, this problem is expressed as minimizing distortion as a function of orthogonal matrix $R \in \mathbb{R}^{d\times d}$ and sub-codebooks $\mathcal{C}^1, \ldots, \mathcal{C}^m \subset \mathbb{R}^{d/m}$ per cell,

$$\begin{array}{ll} \text{minimize} & \sum_{\mathbf{z}\in\mathcal{Z}} \min_{\hat{\mathbf{c}}\in\hat{\mathcal{C}}} \|\mathbf{z} - R\hat{\mathbf{c}}\|^2 \\ \text{subject to} & \hat{\mathcal{C}} = \mathcal{C}^1 \times \cdots \times \mathcal{C}^m \\ & R^{\mathrm{T}}R = I, \end{array} \tag{10}$$

where $|\mathcal{C}^j| = k$ for $j \in \mathcal{M} = \{1, \ldots, m\}$. Given solution $R, \mathcal{C}^1, \ldots, \mathcal{C}^m$, codebook $\mathcal{C}$ is found by (5). For $j \in \mathcal{M}$, sub-codebook $\mathcal{C}^j$ determines a sub-quantizer q^j by

$$\mathbf{x} \mapsto q^j(\mathbf{x}) = \arg\min_{\hat{\mathbf{c}}^j\in\mathcal{C}^j} \|\mathbf{x} - \hat{\mathbf{c}}^j\| \tag{11}$$

for $\mathbf{x} \in \mathbb{R}^{d/m}$, as in (2); collectively, sub-quantizers determine a product quantizer $q = (q^1, \ldots, q^m)$ by (4). Local optimization can then be seen as a mapping $\mathcal{Z} \mapsto (R, q)$. Following [9, 15], there are two solutions that we briefly describe here, focusing more on OPQ_{P}.

Parametric solution (OPQ_{P} [9]) is the outcome of assuming a d-dimensional, zero-mean normal distribution $\mathcal{N}(\mathbf{0}, \Sigma)$ of residual data $\mathcal{Z}$ and minimizing the theoretical lower distortion bound as a function of R alone [9]. That is, R is optimized independently prior to codebook optimization, which can follow by independent k-means per subspace, exactly as in PQ.

Given the $d \times d$ positive definite covariance matrix Σ, empirically measured on $\mathcal{Z}$, the solution for R is found in closed form, in two steps. First, rotating data by $\hat{\mathbf{z}} \leftarrow R^{\mathrm{T}}\mathbf{z}$ for $\mathbf{z} \in \mathcal{Z}$ should yield a block-diagonal covariance matrix $\hat{\Sigma}$, with the j-th diagonal block being sub-matrix $\hat{\Sigma}_{jj}$ of j-th subspace, for $j \in \mathcal{M}$. That is, subspace distributions should be pairwise *independent*. This is accomplished *e.g.* by diagonalizing Σ as $U\Lambda U^{\mathrm{T}}$.

Second, determinants $|\hat{\Sigma}_{jj}|$ should be equal for $j \in \mathcal{M}$, *i.e.*, variance should be *balanced* across subspaces. This is achieved by *eigenvalue allocation* [9]. In particular, a set $\mathcal{B}$ of m buckets B^j is initialized with $B^j = \emptyset$, $j \in \mathcal{M}$, each of capacity $d^* = d/m$. Eigenvalues in Λ are then traversed in descending order, $\lambda_1 \geq \cdots \geq \lambda_d$. Each eigenvalue λ_s, $s = 1, \ldots, d$, is greedily allocated to the non-full bucket B^* of minimal variance, *i.e.*, $B^* \leftarrow B^* \cup s$ with

$$B^* = \arg\min_{\substack{B\in\mathcal{B} \\ |B|<d^*}} \prod_{s\in B} \lambda_s, \tag{12}$$

until all buckets are full. Then, buckets determine a re-ordering of dimensions: if vector $\mathbf{b}^j \in \mathbb{R}^{d^*}$ contains elements of bucket B^j (in any order) for $j \in \mathcal{M}$ and $\mathbf{b} = (\mathbf{b}^1, \ldots, \mathbf{b}^m)$, then vector $\mathbf{b}$ is read off as a permutation π of set $\{1, \ldots, d\}$. If P_π is the permutation matrix of π, then matrix UP_π^{T} represents a re-ordering of eigenvectors of Σ and is the final solution for R. In other words, $\mathcal{Z}$ is first PCA-aligned and then dimensions are grouped in subspaces exactly as eigenvalues are allocated to buckets.

Non-parametric solution (OPQ_{NP} [9] or *Ck-means* [15]) is a variant of k-means, carried out in all m subspaces in parallel, interlacing in each iteration its two traditional steps *assign* and *update* with steps to *rotate* data and optimize R, *i.e.*, *align* centroids to data. OPQ_{P} is extremely faster than OPQ_{NP} in practice. Because we locally optimize thousands of quantizers, OPQ_{NP} training is impractical, so we only use it in one small experiment in section 5.2 and otherwise focus on OPQ_{P}, which we refer to as I-OPQ in the sequel.

4.3. Example

To illustrate the benefit of local optimization, we experiment on our synthetic dataset SYNTH1M, containing 1M 128-dimensional data points and 10K queries, generated by taking 1000 samples from each of 1000 components of an anisotropic Gaussian mixture distribution. All methods are non-exhaustive as in section 4.1, *i.e.* using a coarse quantizer, inverted lists and PQ-encoded residuals; however, all optimization variants are global except for LOPQ. For fair comparison here and in section 5, I-OPQ is our own non-exhaustive adaptation of [9]. IVFADC (PQ) [12] uses natural order of dimensions and no optimization.

Figure 2 shows results on ANN search. On this extremely multi-modal distribution, I-OPQ fails to improve

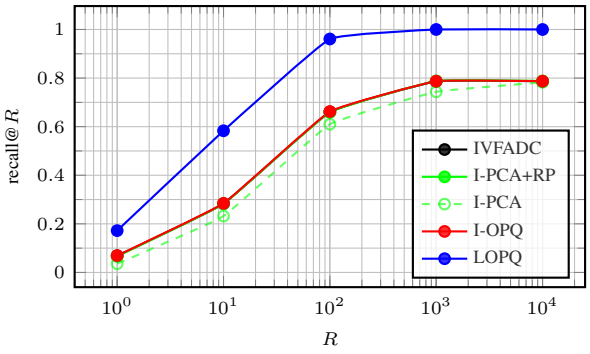

Figure 2. Recall@R performance on SYNTH1M—recall@R is defined in section 5.1. We use $K = 1024$ and $w = 8$ for all methods; for all product quantizers, we use $m = 8$ and $k = 256$. Curves for IVFADC, I-OPQ and I-PCA+RP coincide everywhere.

over IVFADC. PCA-aligning all data and allocating dimensions in decreasing order of eigenvalues is referred to as I-PCA. This is even worse than natural order, because *e.g.* the largest d/m eigenvalues are allocated in a single subspace, contrary to the balancing objective of I-OPQ. Randomly permuting dimensions after global PCA-alignment, referred to as I-PCA+RP, alleviates this problem. LOPQ outperforms all methods by up to 30%.

4.4. Searching on a multi-index

The case of a second-order multi-index is less trivial, as the space overhead is prohibitive to locally optimize per cell as in section 4.1. Hence, we separately optimize per cell of the two subspace quantizers and encode two sub-residuals. We call this *product optimization*, or Multi-LOPQ.

Product optimization. Two subspace quantizers Q^1, Q^2 of K centroids each are built as in [3], with associated codebooks $\mathcal{E}^j = \{\mathbf{e}^j_1, \dots, \mathbf{e}^j_K\}$ for $j = 1, 2$. Each data point $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2) \in \mathcal{X}$ is quantized to cell $Q(\mathbf{x}) = (Q^1(\mathbf{x}^1), Q^2(\mathbf{x}^2))$. An inverted list $\mathcal{L}_{i^1 i^2}$ is kept for each cell $(\mathbf{e}^1_{i^1}, \mathbf{e}^2_{i^2})$ on grid $\mathcal{E} = \mathcal{E}^1 \times \mathcal{E}^2$, for $i^1, i^2 \in \mathcal{K}$.

At the same time, Q^1, Q^2 are employed for residuals as well, as in Multi-D-ADC [3]. That is, for each data point $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2) \in \mathcal{X}$, residuals $\mathbf{x}^j - Q^j(\mathbf{x}^j)$ for $j = 1, 2$ are PQ-encoded. However, because the codebook induced on $\mathbb{R}^d$ by Q^1, Q^2 is extremely fine (K^2 cells on the grid), locally optimizing per cell is not an option—the total space overhead *e.g.* would be $O((d^2 + dk)K^2)$. What we do is separately optimize per subspace: similarly to (9), let

$$\mathcal{Z}^j_i = \{\mathbf{x}^j - \mathbf{e}^j_i : \mathbf{x} \in \mathcal{X},\ Q^j(\mathbf{x}^j) = \mathbf{e}^j_i\}. \qquad (13)$$

contain the residuals of points $\mathbf{x} \in \mathcal{X}$ whose j-th sub-vector is quantized to cell $\mathbf{e}^j_i$ for $i \in \mathcal{K}$ and $j = 1, 2$. We then locally optimize each set $\mathcal{Z}^j_i$ as in discussed section 4.2, yielding a rotation matrix R^j_i and a product quantizer q^j_i.

Now, given a point $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2) \in \mathcal{X}$ quantized to cell $(\mathbf{e}^1_{i^1}, \mathbf{e}^2_{i^2}) \in \mathcal{E}$, its sub-residuals $\mathbf{z}^j = \mathbf{x}^j - \mathbf{e}^j_{i^j}$ are rotated and PQ-encoded as $q^j_{i^j}(\hat{\mathbf{z}}^j) = q^j_{i^j}((R^j_{i^j})^{\mathrm{T}}\mathbf{z}^j)$ for $j = 1, 2$. That is, encoding is separately adjusted per sub-centroid i^1 (resp., i^2) in the first (resp., second) subspace.

Given a query $\mathbf{y}$, rotations $\hat{\mathbf{y}}^j_{i^j} = (R^j_{i^j})^{\mathrm{T}}(\mathbf{y}^j - \mathbf{e}^j_{i^j})$ are lazy-evaluated for $i^j = 1, ..., K$ and $j = 1, 2$, *i.e.* computed on demand by multi-sequence and stored for re-use. For each point index p fetched in cell $(\mathbf{e}^1_{i^1}, \mathbf{e}^2_{i^2}) \in \mathcal{E}$ with associated residuals $\hat{\mathbf{z}}^j_p$ for $j = 1, 2$, asymmetric distance

$$\|\hat{\mathbf{y}}^1_{i^1} - q^1_{i^1}(\hat{\mathbf{z}}^1_p)\|^2 + \|\hat{\mathbf{y}}^2_{i^2} - q^2_{i^2}(\hat{\mathbf{z}}^2_p)\|^2 \qquad (14)$$

is computed. Points are ranked according to this distance.

When considering the entire space $\mathbb{R}^d$, this kind of optimization is indeed local per cell, but more constrained than in section 4.2. For instance, the effective rotation matrix in cell $(\mathbf{e}^1_{i^1}, \mathbf{e}^2_{i^2}) \in \mathcal{E}$ is constrained to be block-diagonal with blocks $R^1_{i^1}, R^2_{i^2}$, keeping rotations within-subspace. By contrast, OMulti-D-OADC [8] employs an arbitrary rotation matrix that is however fixed for all cells.

Analysis. Comparing to Multi-D-ADC [3], the space overhead remains (asymptotically) the same as in section 4.1, *i.e.*, $O(K(d^2 + dk))$. The query time overhead is $O(Kd^2)$ in the worst case, but much lower in practice.

5. Experiments

5.1. Experimental setup

Datasets. We conduct experiments on four publicly available datasets. Three of them are popular in state-of-the-art ANN methods: SIFT1M, GIST1M [12] and SIFT1B [13][1]. SIFT1M dataset contains 1 million 128-dimensional SIFT vectors and 10K query vectors; GIST1M contains 1 million 960-dimensional GIST vectors and 1000 query vectors; SIFT1B contains 1 billion SIFT vectors and 10K queries.

Given that LOPQ is effective on multi-modal distributions, we further experiment on MNIST[2] apart from our synthetic dataset SYNTH1M discussed in section 4.3. MNIST contains 70K images of handwritten digits, each represented as a 784-dimensional vector of raw pixel intensities. As in [9, 8], we randomly sample 1000 vectors as queries and use the remaining as the data.

Evaluation. As in related literature [12, 9, 13, 3, 16, 15], we measure search performance via the *recall@R* measure, *i.e.* the proportion of queries having their nearest neighbor ranked in the first R positions. Alternatively, recall@R is the fraction of queries for which the nearest neighbor would be correctly found if we verified the R top-ranking vectors using exact Euclidean distances. Recall@1 is the most important, and is equivalent to the *precision* of [14].

Re-ranking. Following [13], second-order residuals can be used for re-ranking along with LOPQ variants, but for fair comparison we only apply it with a single index. This new variant, LOPQ+R, locally optimizes second-order subquantizers per cell. However, rotation of second-order residuals is only optimized globally; otherwise there would be an additional query time overhead on top of [13].

Settings. We always perform search in a non-exhaustive manner, either with a single or a multi-index. In all cases, we use $k = 256$, *i.e.* 8 bits per sub-quantizer. Unless otherwise stated, we use 64-bit codes produced with $m = 8$. On SIFT1B we also use 128-bit codes produced with $m = 16$, except when re-ranking, where $m = m' = 8$ is used instead, as in [13]. For all multi-index methods, T refers to the target number of points fetched by multi-sequence.

[1] http://corpus-texmex.irisa.fr/
[2] http://yann.lecun.com/exdb/mnist/

Compared methods (MNIST, SIFT1M, GIST1M). We compare against three of the methods discussed in section 4.3, all using a single index on a coarse quantizer and PQ-encoded residuals, with any optimization being global. In particular, IVFADC [12], our I-PCA+RP, and our non-exhaustive adaptation of OPQ [9], using either OPQ_{P} or OPQ_{NP} global optimization. These non-exhaustive variants are not only faster, but also superior. OPQ_{NP} is too slow to train, so is only shown for MNIST; otherwise I-OPQ refers to OPQ_{P}. We do not consider transform coding [4] or ITQ [10] since they are outperformed by I-OPQ in [9].

Compared methods (SIFT1B). After some experiments on a single index comparing mainly to IVFADC and I-OPQ, we focus on using a multi-index, comparing against Multi-D-ADC [3] and its recent variant OMulti-D-OADC [8], currently the state-of-the-art. Both methods PQ-encode the residuals of the subspace quantizers. Additionally, OMulti-D-OADC uses OPQ_{NP} to globally optimize both the initial data prior to multi-index construction and the residuals. We also report results for IVFADC with re-ranking (IVFADC+R) [13], Ck-means [15], KLSH-ADC [17], multi-index hashing (Multi-I-Hashing) [16], and the very recent joint inverted indexing (Joint-ADC) [19].

Implementation. Results followed by a citation are reproduced from the corresponding publication. For the rest we use our own implementations in Matlab and C++ on a 8-core machine with 64GB RAM. For k-means and exhaustive nearest neighbor assignment we use `yael`[3].

5.2. Results on MNIST, SIFT1M, GIST1M

MNIST is considered first. This is the only case where we report results for OPQ_{NP}, since it is favored over OPQ_{P} in [9], and MNIST is small enough to allow for training. As suggested in [9], we run 100 iterations for OPQ_{NP} using the implementation provided by the authors.

Recall and distortion results are shown in Figure 3. Observe how the gain of OPQ_{NP} over OPQ_{P} is very limited now that global optimization is performed on the residuals. This can be explained by the fact that residuals are expected to follow a rather unimodal distribution, hence closer to the Gaussian assumption of OPQ_{P}. The performance of our simplified variant I-PCA+RP is very close to I-OPQ. Still, separately optimizing the residual distribution of each cell gives LOPQ a significant gain over all methods.

SIFT1M and GIST1M results are shown in Figures 4 and 5 respectively, only now OPQ is limited to OPQ_{P}. As in [9], we use the optimal dimension order for each dataset for baseline method IVFADC [12], *i.e.* natural (resp. structured) order for SIFT1M (resp. GIST1M). In both cases, LOPQ clearly outperforms all globally optimized approaches. For SIFT1M its gain at $R = 1, 10$ is more than

[3] https://gforge.inria.fr/projects/yael

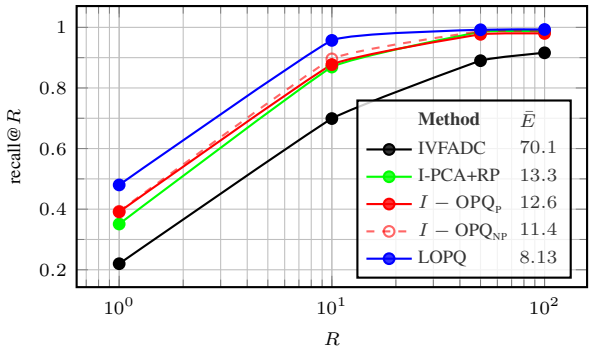

Figure 3. Recall@R on MNIST with $K = 64$, found to be optimal, and $w = 8$. $\bar{E} = E/n$: average distortion per point.

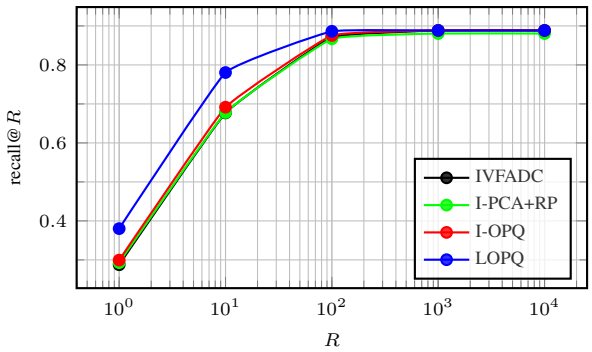

Figure 4. Recall@R on SIFT1M with $K = 1024$, $w = 8$.

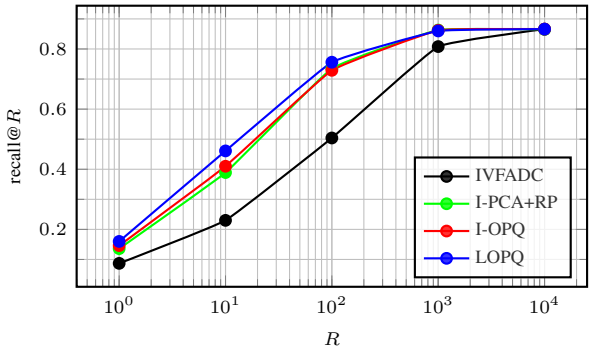

Figure 5. Recall@R on GIST1M with $K = 1024$, $w = 16$.

8% over I-OPQ, which is close to the baseline. The gain is lower for GIST1M but still 5% for $R = 10$. This is where I-OPQ improves most, in agreement with [9, 8], so LOPQ has less to improve. This is attributed to GIST1M mostly being subject to one Gaussian distribution in [8]. I-PCA+RP is always slightly below I-OPQ.

Figures 6 and 7 plot recall@10 versus bit allocation per point (through varying m) and soft assignment neighborhood w, respectively. LOPQ enjoys superior performance in all cases, with the gain increasing with lower bit rates and more soft assignment. The latter suggests more precise distance measurements, hence lower distortion.

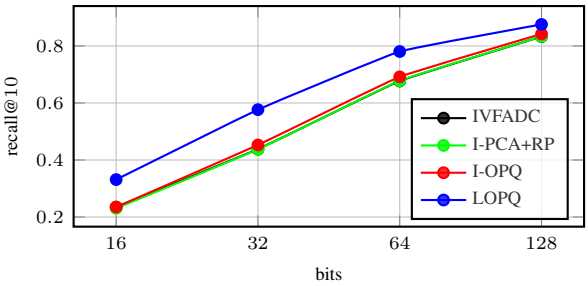

Figure 6. Recall@10 on SIFT1M versus bit allocation per point, with $K = 1024$ and $w = 8$. For $16, 32, 64$ and 128 bits, m is respectively $2, 4, 8$ and 16.

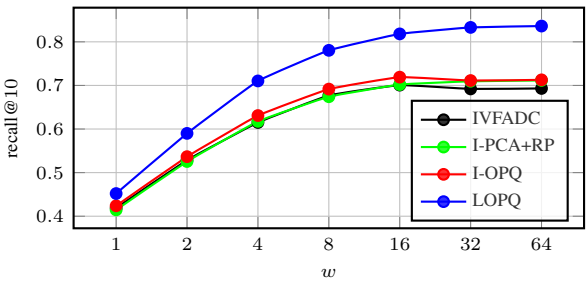

Figure 7. Recall@10 on SIFT1M versus w, with $K = 1024$ and $m = 8$.

Method	$R = 1$	$R = 10$	$R = 100$
Ck-means [15]	–	–	0.649
IVFADC	0.106	0.379	0.748
IVFADC [13]	0.088	0.372	0.733
I-OPQ	0.114	0.399	0.777
Multi-D-ADC [3]	0.165	0.517	0.860
LOR+PQ	0.183	0.565	0.889
LOPQ	**0.199**	**0.586**	**0.909**

Table 1. Recall@$\{1, 10, 100\}$ on SIFT1B with 64-bit codes, $K = 2^{13} = 8192$ and $w = 64$. For Multi-D-ADC, $K = 2^{14}$ and $T = 100$K. Rows including citations reproduce authors' results.

5.3. Results on SIFT1B

64-bit code ($m = 8$) results are shown in Table 1, including I-OPQ, Ck-means [15], Multi-D-ADC [3] and IVFADC without re-ranking, since re-ranking does not improve performance at this bit rate [13]. All methods are using a single index except Multi-D-ADC that uses a multi-index and Ck-means that is exhaustive. For IVFADC we both reproduce results of [13] and report on our re-implementation. To illustrate the individual gain from locally optimized rotation and sub-quantizers, we also include our sub-optimal variant LOR+PQ as discussed in section 4.1. Both LOR+PQ and LOPQ are clearly superior to all methods, with a gain of 18% over I-OPQ and 7% over Multi-D-ADC for recall@10, although the latter is using a multi-index.

T	Method	$R = 1$	10	100
	Multi-I-Hashing [16]	–	–	0.420
	KLSH-ADC [17]	–	–	0.894
	Joint-ADC [19]	–	–	0.938
20K	IVFADC+R [13]	0.262	0.701	0.962
	LOPQ+R	**0.350**	**0.820**	**0.978**
10K	Multi-D-ADC [3]	0.304	0.665	0.740
	OMulti-D-OADC [8]	0.345	0.725	**0.794**
	Multi-LOPQ	**0.430**	**0.761**	0.782
30K	Multi-D-ADC [3]	0.328	0.757	0.885
	OMulti-D-OADC [8]	0.366	0.807	**0.913**
	Multi-LOPQ	**0.463**	**0.865**	0.905
100K	Multi-D-ADC [3]	0.334	0.793	0.959
	OMulti-D-OADC [8]	0.373	0.841	**0.973**
	Multi-LOPQ	**0.476**	**0.919**	**0.973**

Table 2. Recall@$\{1, 10, 100\}$ on SIFT1B with 128-bit codes and $K = 2^{13} = 8192$ (resp. $K = 2^{14}$) for single index (resp. multi-index). For IVFADC+R and LOPQ+R, $m' = 8$, $w = 64$. Results for Joint-ADC and KLSH-ADC are taken from [19]. Rows including citations reproduce authors' results.

128-bit code results are presented in Table 2 and Figure 8, with our solutions including a single index with re-ranking (LOPQ+R) and a multi-index (Multi-LOPQ). Of the re-ranking methods, LOPQ+R has a clear advantage over IVFADC+R, where we adopt $m = m' = 8$ since this option is shown to be superior in [13]. All remaining methods use $m = 16$. Multi-I-Hashing [16], KLSH-ADC [16] and Joint-ADC [19] are all inferior at $R = 100$, although the latter two require 4 times more space.

The current state-of-the-art results come with the use of a multi-index, also boasting lower query times. The recent highly optimized OMulti-D-OADC [8] outperforms Multi-D-ADC [3]. However, the performance of our product optimization Multi-LOPQ is unprecedented, setting a new state-of-the-art on SIFT1B at 128-bit codes and enjoying nearly 10% gain over OMulti-D-OADC on the most important measure of precision (recall@1). Multi-index cells are very fine, hence residuals are lower and local optimization yields lower distortion, although constrained.

5.4. Overhead analysis

Both space and time overhead is constant in data size n.

Space overhead on top of IVFADC (resp. Multi-D-ADC) refers to local rotation matrices and sub-quantizer centroids per cell. For rotation matrices, this is Kd^2 (resp. $2K(d/2)^2$) for single index (resp. multi-index). In practice, this overhead is around 500MB on SIFT1B. For sub-quantizer centroids, overhead is Kdk in all cases. In practice, this is 2GB on SIFT1B for Multi-LOPQ with $K = 2^{14}$. Given that the index space for SIFT1B with 128-bit codes

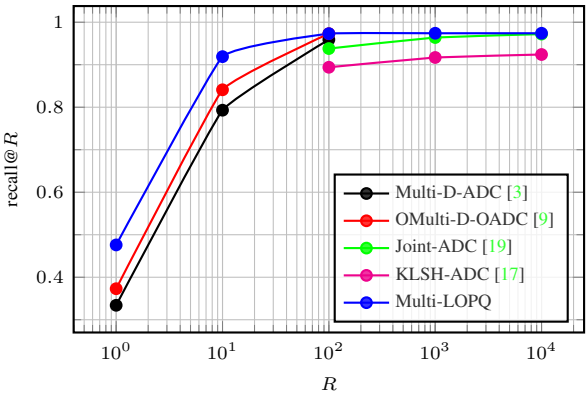

Figure 8. Recall@R on SIFT1B with 128-bit codes and $T = 100$K, following Table 2.

is 21GB in the worst case, this overhead is reasonable.

Query time overhead on top of IVFADC (resp Multi-D-ADC) is the time needed to rotate the query for each soft-assigned cell, which amounts to w ($2K$ worst-case) multiplications of a $d \times d$ ($\frac{d}{2} \times \frac{d}{2}$) matrix and a d ($\frac{d}{2}$)-dimensional vector for single index (resp. multi-index). In practice, the multiplications for multi-index are far less. The average overhead on SIFT1B as measured for Multi-LOPQ is 0.776, 1.92, 4.04ms respectively for $T = 10$K, 30K, 100K.

Multi-D-ADC takes 49ms for $T = 100$K [3], bringing total query time to 53ms. Or, with 7ms for $T = 10$K in [3], Multi-LOPQ outperforms by 5% the previous state-of-the-art 128-bit precision on SIFT1B in less than 8ms.

6. Discussion

Beneath LOPQ lies the very simple idea that no single centroid should be wasted by not representing actual data, but rather each centroid should contribute to lowering distortion. Hence, to take advantage of PQ, one should attempt to use and optimize product quantizers over parts of the data only. This idea fits naturally with a number of recent advances, boosting large scale ANN search beyond the state-of-the-art without significant cost.

It is straightforward to use LOPQ exhaustively as well, by visiting all cells. This of course requires computing K (for LOPQ) or $2K$ (for Multi-LOPQ) lookup tables and rotation matrices instead of just one (*e.g.* for OPQ). However, given the superior performance of residual-based schemes [3, 12], this overhead may still be acceptable. For large scale, exhaustive search is not an option anyway.

LOPQ resembles a two-stage fitting of a *mixture distribution*: component means followed by conditional densities via PQ. Joint optimization of coarse and local quantizers would then seem like a possible next step, but such an attempt still eludes us due to the prohibitive training cost. It would also make sense to investigate the connection to tree-based methods to ultimately compress sets of points as in [1], while at the same time being able to search non-exhaustively without reducing dimensionality.

References

[1] R. Arandjelovic and A. Zisserman. Extremely low bit-rate nearest neighbor search using a set compression tree. Technical report, 2013. 1, 8

[2] Y. Avrithis. Quantize and conquer: A dimensionality-recursive solution to clustering, vector quantization, and image retrieval. In *ICCV*. 2013. 2

[3] A. Babenko and V. Lempitsky. The inverted multi-index. In *CVPR*, 2012. 1, 2, 3, 5, 6, 7, 8

[4] J. Brandt. Transform coding for fast approximate nearest neighbor search in high dimensions. In *CVPR*, 2010. 2, 6

[5] V. Chandrasekhar, Y. Reznik, G. Takacs, D. M. Chen, S. S. Tsai, R. Grzeszczuk, and B. Girod. Compressing feature sets with digital search trees. In *ICCV Workshops*. IEEE, 2011. 1

[6] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262. ACM New York, NY, USA, 2004. 2

[7] J. Delhumeau, P. Gosselin, H. Jegou, and P. Perez. Revisiting the VLAD image representation. In *ACM Multimedia*, Oct 2013. 2

[8] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. Technical report, 2013. 2, 5, 6, 7

[9] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, 2013. 1, 2, 3, 4, 5, 6, 8

[10] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011. 1, 2, 6

[11] K. He, F. Wen, and J. Sun. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In *CVPR*, 2013. 1, 2

[12] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1), 2011. 1, 2, 3, 4, 5, 6, 8

[13] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *ICASSP*, 2011. 3, 5, 6, 7

[14] M. Muja and D. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *ICCV*, 2009. 1, 2, 5

[15] M. Norouzi and D. Fleet. Cartesian k-means. In *CVPR*, 2013. 1, 2, 3, 4, 5, 6, 7

[16] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in Hamming space with multi-index hashing. In *CVPR*, 2012. 1, 2, 5, 6, 7

[17] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: a comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, Aug. 2010. 2, 6, 7, 8

[18] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008. 2

[19] Y. Xia, K. He, F. Wen, and J. Sun. Joint inverted indexing. In *ICCV*, 2013. 2, 6, 7, 8