Revisiting additive quantization

Julieta Martinez Joris Clement Holger H. Hoos James J. Little

University of British Columbia
{julm, jclement, hoos, little}@cs.ubc.ca

Abstract. We revisit Additive Quantization (AQ), an approach to vector quantization that uses multiple, full-dimensional, and non-orthogonal codebooks. Despite its elegant and simple formulation, AQ has failed to achieve state-of-the-art performance on standard retrieval benchmarks, because the encoding problem, which amounts to MAP inference in multiple fully-connected Markov Random Fields (MRFs), has proven to be hard to solve. We demonstrate that the performance of AQ can be improved to surpass the state of the art by leveraging iterated local search, a stochastic local search approach known to work well for a range of NP-hard combinatorial problems. We further show a direct application of our approach to a recent formulation of vector quantization that enforces sparsity of the codebooks. Unlike previous work, which required specialized optimization techniques, our formulation can be plugged directly into state-of-the-art lasso optimizers. This results in a conceptually simple, easily implemented method that outperforms the previous state of the art in solving sparse vector quantization. Our implementation is publicly available.[1]

1 Introduction

Computer vision applications often involve computing the similarity of many high-dimensional, real-valued image representations, in a process known as feature matching. When large databases of images are used, this results in significant computational bottlenecks. For example, in structure from motion [1], it is common to estimate the relative viewpoint of each image in large collections of photographs by computing the pairwise similarity of several million SIFT [2] descriptors; it is also now common for retrieval and classification datasets to comprise millions of images [3, 4].

In practice, the large-scale retrieval problem often translates into large-scale approximate nearest neighbour (ANN) search and has traditionally been addressed with hashing [5, 6]. However, a family of methods based on vector quantization has recently demonstrated superior performance and scalability, sparking interest from the machine learning, computer vision and multimedia retrieval communities [7–12]. These methods are all based on multi-codebook quantization (MCQ), a generalization of k-means clustering with cluster centres arising from the sum of entries in multiple codebooks. Other applications of MCQ include large-scale maximum inner product search (MIPS) [13, 14], and the compression of deep neural networks for mobile devices [15].

Like k-means clustering, MCQ is posed as the search for a set of codes and codebooks that best approximate a given dataset. While early approaches to MCQ were

[1] https://github.com/una-dinosauria/local-search-quantization

designed enforcing codebook orthogonality [9, 10], more recent methods make use of non-orthogonal, often full-dimensional codebooks [7, 8, 11]. A problem faced by these methods is that encoding, in general, becomes NP-hard. Moreover, encoding is to be performed on large databases and must therefore often be carried out under very tight time constraints.

We note that the combinatorial optimization community has been dealing with similar problems for many years, and competitions to solve NP-hard problems as efficiently as possible (*e.g.* the SAT competition series [16]) have driven research on fast combinatorial optimization methods such as stochastic local search [17] (SLS) and portfolio-based solvers [18]. Inspired by the combinatorial optimization literature, our main contribution is the introduction of an SLS-based algorithm that achieves low quantization error and high encoding speed in MCQ. We also discuss a series of implementation details that make our algorithm fast in practice and demonstrate an application of our approach that incorporates sparsity constraints in the codebooks.

2 Related work

First, we introduce some notation, following mostly Norouzi and Fleet [19]. Formally, we denote the set to quantize as $X \in \mathbb{R}^{d \times n}$, having n data points with d dimensions each; MCQ is the problem of finding m codebooks $C_i \in \mathbb{R}^{d \times h}$ and the corresponding codes B_i that minimize quantization error, *i.e.*, to determine

$$\min_{C_i, B_i} \|X - [C_1, C_2, \ldots, C_m] \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{bmatrix} \|_2^2, \tag{1}$$

where $B_i = [\mathbf{b}_{i1}, \mathbf{b}_{i2}, \ldots, \mathbf{b}_{in}] \in \{0, 1\}^{h \times n}$, and each subcode $\mathbf{b}_i$ is limited to having only one non-zero entry: $\|\mathbf{b}_i\|_0 = 1$, $\|\mathbf{b}_i\|_1 = 1$. Letting $C = [C_1, C_2, \ldots, C_m]$ and $B = [B_1, B_2, \ldots, B_m]^\top$, we can rewrite expression 1 more succinctly as

$$\min_{C,B} \|X - CB\|_2^2. \tag{2}$$

Early work in MCQ can be traced back to the 1980s, when quantization was heavily studied in the context of compressing signals before transmission [20]. Interest in the field was renewed in 2010, when Jégou *et al.* [10] introduced product quantization (PQ), noticing that quantization could be effectively used to search for nearest neighbours in high-dimensional spaces. PQ made a codebook orthogonality assumption that has the advantage of requiring only m table lookups to compute the approximate distance between a query and a compressed database element. Moreover, optimal encoding is easily achieved by solving m d/m-dimensional nearest neighbour problems in small sets of h elements (typically, $h = 256$). It was later shown [9, 19], that a global rotation of the data can also be easily learned, yielding lower quantization error. This approach is often called optimized product quantization (OPQ).

More recently, Babenko and Lempitsky proposed additive quantization (AQ) [7], which drops the orthogonality assumption of PQ and uses full-dimensional codebooks. This formulation corresponds to Expression 2 without further constraints and provides the basis for our work. In spite of achieving superior performance to PQ and OPQ, the authors quickly noticed two main disadvantages of their approach: first, that encoding (finding B) can be expressed as a large number of hard combinatorial problems and becomes a major computational bottleneck of the system; second, that distance computation requires $O(m^2)$ (as opposed to m) table lookups, increasing query time significantly with respect to PQ [7].

In a parallel line of research, Zhang *et al*. proposed composite quantization (CQ) [11], which also relaxes the orthogonality constraint of PQ and optimizes for codebooks with constant inner products: $\langle C_i^\top, C_j \rangle = \xi, \forall\, i, j \neq i \in \{1, 2, \ldots, m\}$ (notice that, in PQ, $\xi = 0$). A crucial advantage of this formulation is that distance computation requires only m table lookups and thus is directly comparable to PQ. As a side effect, the constraint renders the encoding problem easier to solve, and the authors note that using iterated conditional modes (ICM) with 3 iterations obtains "satisfactory results" [11].

The encoding problem in AQ. Our work focuses on improving the encoding time and performance of the AQ formulation – that is, given the data X and codebooks C, we search for a method to find the codes B that minimize Expression 2. Formally, the encoding problem amounts to MAP inference[2] in n fully-connected MRFs with m nodes each, which in the general case is NP-hard. In these MRFs, the minimum energy is achieved by finding the subcodes $\mathbf{b}_i$ that minimize the squared distance between the vector to encode $\mathbf{x}$, and its approximation $\hat{\mathbf{x}}$:

$$\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \|\mathbf{x} - \sum_i^m C_i \mathbf{b}_i\|_2^2 = \|\mathbf{x}\|_2^2 - 2 \cdot \sum_i^m \langle \mathbf{x}, C_i \mathbf{b}_i \rangle + \|\sum_i^m C_i \mathbf{b}_i\|_2^2 \tag{3}$$

where the norm of the approximation $\|\hat{\mathbf{x}}\|_2^2 = \|\sum_i^m C_i \mathbf{b}_i\|_2^2$ can be expanded as

$$\|\sum_i^m C_i \mathbf{b}_i\|_2^2 = \sum_i^m \|C_i \mathbf{b}_i\|_2^2 + \sum_i^m \sum_{j \neq i}^m \langle C_i \mathbf{b}_i, C_j \mathbf{b}_j \rangle. \tag{4}$$

Posed as an MRF with m nodes, the $\|\mathbf{x}\|_2^2$ term in (3) can be discarded because it is a constant with respect to $\mathbf{b}_i$; the $-2 \cdot \sum_i^m \langle \mathbf{x}, C_i \mathbf{b}_i \rangle$ and $\sum_i^m \|C_i \mathbf{b}_i\|_2^2$ terms are summed up and become the unary terms, and $\sum_i^m \sum_{j \neq i}^m \langle C_i \mathbf{b}_i, C_j \mathbf{b}_j \rangle$ becomes the pairwise terms. Since each code $\mathbf{b}_i$ may take any value between 1 and h, there are h^m possible configurations for each MRF (typically, $m = \{8, 16\}$, and $h = 256$), which renders the problem inherently combinatorial and, in general, NP-hard [21].

Solving these MRFs is known to be challenging; in fact, the authors of AQ noted that "LBP and ICM, and [...] other algorithms from the MRF optimization library [22] perform poorly" [7]. This has led researchers to resort to expensive construction search methods such as beam search [7] and has since motivated the search for codebook structures where exact encoding is efficient [8, 11].

[2] Unfortunately enough, MAP inference is often called *decoding* due to its historical use in the receiving end of error-correcting codes [20].

Reducing query time in AQ. As mentioned before, AQ involves major computational overhead during query time. This occurs when a new query $\mathbf{q}$ is received and the distance to the encoded vectors $\hat{\mathbf{x}}_i$ has to be estimated. This amounts to evaluating Equation 3: $\|\mathbf{q} - \hat{\mathbf{x}}\|_2^2$, which, as we have seen, requires $O(m^2)$ table lookups for evaluating the norm of the encoded vector $\|\hat{\mathbf{x}}\|_2^2$. Babenko and Lempitsky [7] proposed a simple solution to this problem: use $m - 1$ codebooks to quantize $\mathbf{x}$, and use an extra byte to quantize the norm of the approximation $\|\hat{\mathbf{x}}\|_2^2$. The approximation of the norm is likely to be very good, as we are then using $h = 256$ centroids to quantize a scalar. The downside is that the beam search proposed in AQ becomes very expensive, so in [7], preference was given to a hybrid approach called Additive Product Quantization (APQ). APQ, although more tractable in encoding, still incurs considerable overhead at query time. Promising results for AQ were shown for 64 bit codes on SIFT1M (see AQ-7 in Figure 5 of [7]) but have since been surpassed by CQ [11]. Our work starts from this idea and then focuses on improving the encoding process to improve the performance of AQ beyond the state of the art.

Stochastic local search (SLS) algorithms. Top-performing methods for solving many NP-hard problems have been, at several points in time, variations of stochastic local search (SLS), and continue to define the state of the art for solving prominent NP-hard problems, such as the TSP [23]. Given a candidate solution to a given problem instance, SLS methods iteratively examine and move to neighbouring solutions. A formal treatment of the subject involves defining neighbourhood functions, characterizing problem instances and formally defining local search procedures; while this is beyond the scope of this work, we direct interested readers to [17].

Iterated local search (ILS) algorithms, which form the basis for the algorithm we propose in this work, alternate between perturbing the current solution s (with the goal of escaping local minima) and performing local search starting from the perturbed solution, leading to a new candidate solution, s'. At the end of each local search phase, a so-called acceptance criterion is used to decide whether to continue the search process from s or s'. In many applications of ILS, including ours described in the following, the acceptance criterion simply selects the better of s and s'.

A downside of SLS algorithms (and many other heuristic methods that perform well in practice) is that their theoretical performance has historically proven hard to analyze. Similar to prominent deep learning techniques, SLS methods often perform far better than theory predicts, and thus, research in the area is heavily based on empirical analysis. An attempt to achieve a theoretical breakthrough for our method would be out of the scope of this work, so instead, we focus on the thorough empirical evaluation of its performance on a number of benchmarks with varying sizes, protocols and descriptor types to show the strength of our approach.

3 Iterated local search for AQ encoding

We now introduce our ILS algorithm to optimize B in Expression 2. In addition to the acceptance criterion stated above, our algorithm is defined by (a) a local search proce-

dure, (b) a perturbation method to escape local minima, and (c) an initialization method to create the first solution; we next explain our design choices for these components.

3.1 Local search procedure

As our local search procedure we choose ICM. Although ICM was dismissed in [7] as poorly performant for the encoding problem, the algorithm has been successfully used (for solving an admittedly simpler problem) in CQ [11]. ICM offers two key advantages over other local search algorithms: (i) on the theoretical side, it exhibits very good speed, and, (ii) in practice, it can be implemented in a way that is amenable to caching and vectorization. We discuss both advantages in more detail below.

Complexity analysis of ICM. ICM *iterates* over all the nodes in the given MRF, *conditioning* the current node on the assignments of other nodes, and minimizing (finding the *mode* of) the resulting univariate distribution. In a fully-connected MRF, such as the one in our problem, each node represents a subcode $\mathbf{b}_i$. Thus, ICM cycles through m subcodes, and conditions its value on the other $m-1$ subcodes, adding h terms for each conditioning. This results in a complexity of $O(m^2h)$. Comparing to the beam search procedure of AQ, which has a complexity of $O(mh^2(m+\log mh))$ [8], we can see that for typical values of $m \in \{8, 16\}$ and $h = 256$, a single ICM cycle is much faster than beam search. This suggests that we can afford to run several rounds of ICM, which is necessary for ILS, while keeping the overall encoding time low. In practice, our implementation is 30-50$\times$ faster than beam search in AQ[3].

Cache hits and vectorization of ICM. While this is not true in general, in our special case of interest ICM has a second crucial advantage: it can be programmed in a way that is cache-friendly and easy to vectorize.

In practice, the computational bottleneck of ICM arises in the conditioning step, when the algorithm looks at all the neighbours of the ith node and adds the assignments in those nodes to the current one. A naïve implementation, such as that available from off-the-shelf MRF libraries [22], encodes each data point sequentially, looking up the pairwise terms from $O(m^2)$ different tables of size $h \times h$. This results in a large number of cache-misses, as different tables are loaded into cache for each conditioning.

Our key observation from Equation 3 is that only the unary term $-2\sum_i^m \langle \mathbf{x}, C_i\mathbf{b}_i \rangle$ depends on the vector to encode $\mathbf{x}$. Equivalently, it can be seen that the pairwise terms $\sum_i^m \sum_{j \neq i}^m \langle C_i\mathbf{b}_i, C_j\mathbf{b}_j \rangle$ are the same for all the MRFs that arise in the encoding problem. This means that, during the conditioning step, we can condition all the ith subcodes in the database w.r.t. the jth subcode, loading only one $h \times h$ pairwise lookup table into cache at a time. This results in better cache performance, is easily vectorized, and dramatically speeds up ICM in our case. In practice, this is accomplished by switching the order of the for loops in ICM over the entire (or a large portion of the) database. We call this implementation "batched ICM". Our code is publicly available to facilitate the understanding of these details.

[3] https://github.com/arbabenko/Quantizations

3.2 ILS Perturbation

In each incumbent solution s, we choose k codes to perturb by sampling without replacement from the discrete uniform distribution from 1 to m: $i_k \sim \mathcal{U}\{1, m\}$. We then perturb each selected code uniformly at random by setting it to a value between 1 and h: $\mathbf{b}_{i_k} \sim \mathcal{U}\{1, h\}$. This perturbed solution s' is then used as the starting point for the subsequent local search phase, i.e., invocation of ICM. While simple, this perturbation method is commonly used in the SLS literature [17]. We note that our approach generalizes previous work where ICM was used but no perturbations were applied [7, 11], which corresponds to setting $k = 0$. This method is both effective and very fast in practice: compared to ICM, the time spent in this step is negligible.

3.3 ILS Initialization

We use a simple initialization method, setting all the codes to values between 1 and h uniformly at random: $\mathbf{b}_i \sim \mathcal{U}\{1, h\}\ \forall i \in \{1, 2, \ldots, m\}$. We also experimented with other initialization approaches, such as using FLANN [24] to copy the codes of the nearest neighbour in the training dataset, or using the code that minimizes the binary terms (which are expected to dominate the unary terms for large m). However, we did not observe significant improvements over our random initialization after a few rounds of ILS optimization.

Like AQ, we initialize our C and B by running OPQ, followed by a method similar to OTQ [8], but simplified to assume that the dimension assignments are given by a natural partition of adjacent dimensions.

4 The advantages of a simple formulation: easy sparse codebooks

Our approach, building on top of AQ, benefits from using a simple formulation with no extra constraints (Expression 2). The advantages of a simple formulation are not merely aesthetic; in practice, they result in a straightforward optimization procedure and less overhead for the programmer. Furthermore, a more standard formulation might render the problem more amenable to being solved using state-of-the-art optimizers. We now demonstrate one such use case, by implementing a recent MCQ formulation that enforces sparsity on the codebooks [12].

Motivation for sparse codebooks. Zhang *et al*. [12] motivate the use of sparse codebooks in the context of very large-scale applications, which deal with billions of datapoints and are better suited for search with an inverted-index [25–27]. In this case, the time spent computing the lookup tables becomes non-negligible, which is specially true for methods that use full-dimensional codebooks, such as CQ [11], AQ [7] and, by extension, ours. For example, Zhang *et al*. have demonstrated that enforcing sparsity in the codebooks can lead up to 30% speedups with a state-of-the-art inverted index [27] on the SIFT1B dataset (see Table 2 in [12]). This method is called Sparse CQ (SCQ).

There is a second use case for sparse codebooks. Recently, André *et al*. [28] have demonstrated that PQ scan and other lookup-based sums, such as those in our approach,

can take advantage of vectorization. The authors have shown up to $6\times$ speedups on distance computation, thus emphasizing further the time spent computing distance tables even for datasets with a few million data points, where linear scanning is the preferred search procedure.

Solving the sparsity constraint. Formally, the sparsity constraint on the codebooks amounts to determining

$$\min_{C,B} \|X - CB\|_2^2 \quad \text{s.t.} \quad \|C\|_0 \leq S. \tag{5}$$

Unfortunately, minimizing a quadratic function with an ℓ_0 constraint is non-convex and, in general, hard to solve directly. Thus, the problem is often relaxed the minimize the convex ℓ_1 norm. Our objective then becomes to determine

$$\min_{C,B} \|X - CB\|_2^2 \quad \text{s.t.} \quad \|C\|_1 \leq \lambda. \tag{6}$$

In SCQ [12], the problem becomes even harder, because on top of sparsity, the codebook elements are forced to have constant products. For this reason, SCQ uses an ad-hoc soft-thresholding algorithm to solve for C. Our problem is simpler, because we do not have to satisfy the inter-codebook constraint and can be posed as an ℓ_1-regularized least-squares problem. To achieve this, we rewrite Expression 6 as

$$\min_{C,B} \|B^\top C^\top - X^\top\|_2^2 \quad \text{s.t.} \quad \|C\|_1 \leq \lambda, \tag{7}$$

and it becomes apparent that the approximation of the ith column of $X^\top$ depends only on product of $B^\top$ and the ith column of $C^\top$. Thus, we can rewrite Expression 7 as

$$\min_{C,B} \|\hat{B}\hat{\mathbf{c}} - \hat{\mathbf{x}}\|_2^2 \quad \text{s.t.} \quad \|\hat{\mathbf{c}}\|_1 \leq \lambda, \tag{8}$$

where

$$\hat{B} = \begin{bmatrix} B_{(1)}^\top & 0 & \dots & 0 \\ 0 & B_{(2)}^\top & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & B_{(d)}^\top \end{bmatrix}, \hat{\mathbf{c}} = \begin{bmatrix} \mathbf{c}_1^\top \\ \mathbf{c}_2^\top \\ \vdots \\ \mathbf{c}_d^\top \end{bmatrix}, \text{ and } \hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_d^\top \end{bmatrix}. \tag{9}$$

Here, $B_{(i)}^\top$ is the ith copy of $B^\top$, $\mathbf{c}_i$ is the ith row of C, and $\mathbf{x}_i$ is the ith row of X. This formulation corresponds to the well-known lasso. Nearly two decades of research in the lasso have produced many robust and scalable off-the-shelf optimization routines for this problem. Our approach, as opposed to previous work, lacks extra constraints and thus can directly take advantage of such procedures with little overhead to the programmer. For example, it took us less than an hour to integrate the SPGL1 solver by van den Berg and Friedlander [29] into our pipeline. This solver has the additional advantage of not requiring an explicit representation of $\hat{B}$, but can instead be given a function that evaluates to $\hat{B}\hat{\mathbf{c}}$ – this can be implemented as a `for` loop, so we only need to store one copy of the codes $B^\top$ in memory. Note that SPGL1 is only used to find the codebooks C; finding the codes B is still done using our previously described ILS procedure.

5 Experimental setup

Evaluation protocol. We follow previous work and evaluate the performance of our system with recall@N [7–11]. Recall@N produces a monotonically increasing curve from 1 to N representing the empirical probability, computed over the query set, that the N estimated nearest neighbours include the actual nearest neighbour in the database. The goal is obtain the highest possible recall for given N. In information retrieval, recall@1 is considered the most important number on this curve. Also in line with previous work [7–12], in all our experiments, the codebooks have $h = 256$ elements, and we show results using 64 and 128 bits for code length.

We compute approximate squared Euclidean distances applying the expansion of Equation 3, and we use only 7 and 15 codebooks to store codebook indices, while the last 8 bits are dedicated to quantize the (scalar) squared norm of each database entry. In all our experiments, we run our method for 100 iterations and use asymmetric distance, *i.e.*, the distance tables are computed for each query, as in all our baselines.

Baselines. We compare our approach to previous work, controlling for two critical factors in large-scale retrieval: code length and query time. To control for code length, we use subcodebooks with $h = 256$ entries and produce final codes of 64 and 128 bits. To control for query time, we restrict our comparison to methods that require $m = \{8, 16\}$ table lookups to compute approximate distances. Thus, we compare against PQ [10], OPQ [9] and CQ [11], as well as the AQ-7 method presented in [7] (*i.e.*, the original formulation of AQ that we are building on). For PQ and OPQ, we use the publicly available code of Mohammad Norouzi[4], and we reproduce the results reported on the original papers introducing CQ [11] and AQ [7]. We compare our sparse extension against SCQ [12], which is to our knowledge the only paper on the subject.

Another baseline that we could compare against is the recently introduced Optimized Tree Quantization (OTQ) [8]. OTQ learns a tree structure of the codebooks where encoding can be performed exactly using dynamic programming, and has demonstrated competitive results on SIFT1M. The method, however, requires $2m - 1$ table lookups to compute approximate distances during query time, so it does not fit the query time criterion that we are controlling for in our experiments.

Datasets. We tested our method on 6 datasets. Three of these, SIFT1M, SIFT10M and SIFT1B [10], consist of 128-dimensional gradient-based, hand-crafted SIFT features. We also collected a dataset of 128-dimensional deep features using the CNN-M-128 network provided by Chatfield *et al.* [30], computing the features from a central 224×224 crop of the 1.3M images of the ILSRVC-2012 dataset and then subsampling uniformly at random from all classes. It is known that deep features can be effectively used as descriptors for image retrieval [31, 32], and Chatfield *et al.* have shown that this intra-net compression results in a minimal loss of performance [30]. SIFT1M and Convnet1M both have 100 000 training vectors, 10 000 query vectors and 1 million database vectors. SIFT1B has 100 million vectors for training, and 1 billion vectors for base, as

[4] https://github.com/norouzi/ckmeans

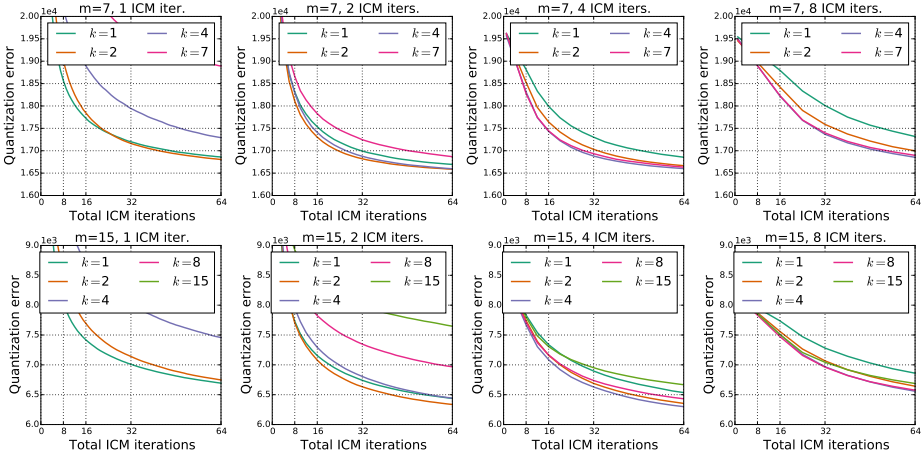

Fig. 1. Quantization error as a function of ILS iterations, ICM iterations and number of codes perturbed k on 10 000 vectors of the SIFT1M train dataset after random initialization. The number of ICM iterations increases in 1, 2, 4, and 8 to the right, and we plot the quantization error for 64 ICM iterations in total – thus, the plots are comparable in amount of computation. Using 4 perturbations and 4 ICM iterations gives good results in all cases, so we use those parameters in all our experiments. Using no perturbations ($k = 0$), as done in previous work [7, 11, 12], leads to values that are above all the plots that we are showing, and stagnates after about 3 ICM iterations.

well as 10 000 queries. On SIFT1B, we followed previous work [9, 11] and used only the first 1 million vectors for training. Better results on SIFT1B can be obtained using an inverted index, but we did not implement this data structure as we focus on improving encoding performance. This also has the added benefit of making our results directly comparable to those shown in CQ [11] and OPQ [19]. With SIFT10M, we followed the same protocol, but only the 10 million first vectors of SIFT1B as base.

We also use 2 datasets where CQ, our closest competitor, was benchmarked: MNIST and LabelMe22K [33]. MNIST is 784-dimensional, and has 10 000 vectors for query and 60 000 vectors for base. LabelMe22K is 512-dimensional and has 2 000 vectors for query and 20 019 vectors for base.

Different datasets have different partitions, and this leads to important differences in the way learning and encoding are performed. The classical datasets SIFT1M, SIFT10M and SIFT1B [10], as well as our Convnet1M dataset have three partitions: train, query and database. In this case, the protocol is to first learn the codebooks using only the train set, then use the learned codebooks to encode the database, and finally evaluate recall@N of the queries w.r.t. the database. We refer to this partition as train/query/base.

However, several earlier studies used only two partitions of the data: query and database. In this case, the iterative codebook learning procedure is run directly on the database, and recall@N is evaluated on the queries w.r.t. the database thereafter. For example, Locally Optimized PQ [34] was evaluated using this partition on SIFT1B by simply ignoring the training set, and CQ was evaluated on MNIST and LabelMe22K using the train/test partitions that the datasets provide for classification [11]. It has also

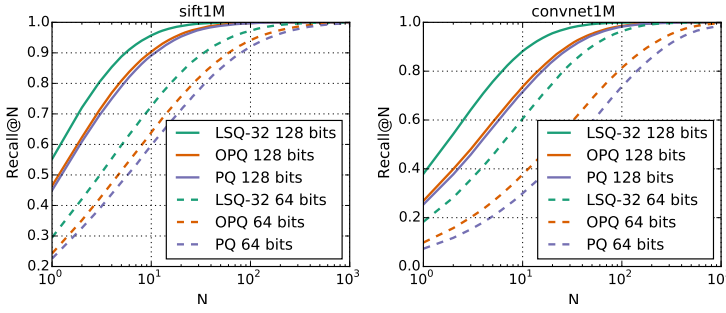

Fig. 2. Recall@N curves for (left) the SIFT1M, and (right) Convnet1M datasets.

	SIFT1M – 64 bits		
	R@1	R@10	R@100
PQ	22.53 ± 0.31	60.14 ± 0.41	91.99 ± 0.17
OPQ	24.34 ± 0.52	63.89 ± 0.30	94.04 ± 0.08
AQ-7 [7]	26	70	95
CQ [11]	29	71	96
LSQ-16	$\underline{29.37} \pm 0.18$	$\underline{72.54} \pm 0.26$	$\underline{97.27} \pm 0.14$
LSQ-32	$\mathbf{29.79} \pm 0.26$	$\mathbf{73.12} \pm 0.20$	$\mathbf{97.49} \pm 0.09$

	SIFT1M – 128 bits		
	R@1	R@2	R@5
PQ	44.62 ± 0.47	60.54 ± 0.61	78.88 ± 0.30
OPQ	46.05 ± 0.25	62.05 ± 0.21	80.59 ± 0.31
AQ-15 [7]	–	–	–
CQ [11]	54	71	88
LSQ-16	$\underline{54.47} \pm 0.37$	$\underline{71.74} \pm 0.33$	$\underline{88.21} \pm 0.48$
LSQ-32	$\mathbf{55.28} \pm 0.21$	$\mathbf{72.26} \pm 0.36$	$\mathbf{88.93} \pm 0.14$

Table 1. Detailed recall@N values for our method on the SIFT1M dataset.

been argued that this partition is better suited for learning inverted indices on very large-scale datasets (see the last paragraph of [27]). We refer to this partition as query/base.

Parameter settings. Our approach needs to set the number of ILS iterations (*i.e.*, the number of times a solution is perturbed and local search is done). At the same time, ICM may cycle through the nodes a number of times, which we call *ICM iterations*. Finally k, the number of elements to perturb, also needs to be defined.

We chose our parameters using a held-out subset of the training set of SIFT1M, keeping the values that minimize quantization error. Figure 1 shows the results of our parameter search on 10 000 SIFT descriptors. We note that, given the same amount of ICM cycles, using 4 ICM iterations and perturbing $k = 4$ code elements results in good performance. We observed similar results on other descriptor types, so we used these values in all our experiments. As shown in Figure 1, the performance of our system depends on the number of ILS iterations used, trading-off computation for recall. We evaluated our method using 16 and 32 ILS iterations on the base set of train/query/base datasets, and refer to these methods as LSQ-$\{16, 32\}$. During training, we used only 8 ILS iterations.

Implementation details and reproducible research. We implemented all our code in Julia [35], a recent high-level language for scientific computing, making use of the `@inbounds` and `@simd` macros in performance-critical parts of the code. All reported running times were measured using a single core on a computer with 64 GB of RAM

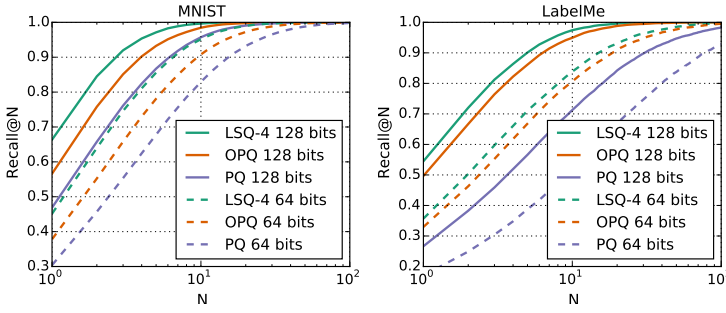

Fig. 3. Recall@N curves for (left) the MNIST, and (right) LabelMe22K datasets.

	MNIST – 64 bits		
	R@1	R@2	R@5
PQ	30.39 ± 0.28	45.70 ± 0.39	67.79 ± 0.46
OPQ	37.81 ± 0.63	55.23 ± 0.44	78.10 ± 0.46
CQ [11]	44	63	84
LSQ-4	$\mathbf{45.13} \pm 0.50$	$\mathbf{63.98} \pm 0.70$	$\mathbf{85.58} \pm 0.40$

	LabelMe22K – 64 bits		
	R@1	R@2	R@5
PQ	17.05 ± 0.53	24.90 ± 0.48	38.78 ± 1.19
OPQ	32.96 ± 0.51	46.29 ± 0.68	66.80 ± 0.55
CQ [11]	35	**51**	71
LSQ-4	$\mathbf{35.69} \pm 1.10$	50.32 ± 1.46	$\mathbf{71.05} \pm 1.31$

Table 2. Detailed recall@N values for our method on the MNIST and LabelMe22K datasets.

and an Intel i7-3930K CPU, which runs at 3.2 GHz and has 12 MB of cache. To render our results reproducible, all our code and data are publicly available.

6 Results

Since our method relies heavily on randomness for encoding, it is natural to think that the final performance of the system could exhibit large variance. To quantify this effect, we ran our method 5 times on each dataset, and report the mean and standard deviation in recall@N achieved by our method. To see how this compares to previous work, we ran PQ and OPQ[5] 5 times (as for our method), and report their mean and standard deviation in recall@N as well. We observe that LSQ, despite relying heavily on randomness for encoding, exhibits a variability in recall similar to that of PQ and OPQ (and presumably that of other baselines as well).

This is consistent with the fact that LSQ solves a large number of independent instances of combinatorial optimisation problems of similar difficulty; in situations like this, the solution qualities achieved within a fixed running time are typically normally distributed ([17], Ch.4). In other words, despite being heavily randomized, the performance of our system turns out to be very stable in practice, because it is averaged over a large number of data points.

[5] PQ has randomness in the k-means initialization, and the OPQ code by Norouzi & Fleet chooses a random initial set of cluster centers.

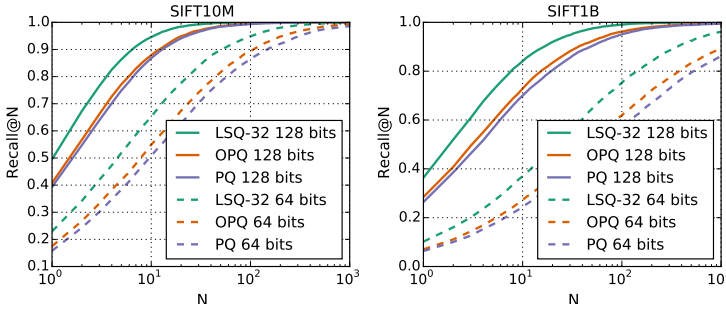

Fig. 4. Recall@N curves for very large-scale datasets: (left) the SIFT10M, and (right) SIFT1B.

Small training/query/base datasets. First, we report the recall@N results for SIFT1M and Convnet1M in Figure 2, where it is immediately clear that LSQ outperforms the classical baselines, PQ and OPQ in recall@N for all values of N. Similarly, our method outperforms CQ when using 16 ILS iterations, and the gap is widened when using 32 iterations. As we will see, analogous effects are observed throughout our results. In Table 1 we show our results in more detail, providing mean recall@N values and standard deviations for the SIFT1M dataset.

On Convnet1M, the advantage of LSQ over PQ/OPQ is more pronounced. When using 64 bits, our method achieves a recall of 18.64, more than double that of PQ at 7.13, and with an 81% improvement over OPQ at 10.28. These results show an increased advantage of our method over orthogonal approaches like PQ/OPQ in deep-learned features, which are expected to dominate computer vision applications in coming years.

Query/base datasets. In Figure 3, we report results on datasets with query/base partitions: MNIST and LabelMe22K. Despite requiring less computation in these datasets (only 4 ILS iterations, instead of 16-32), our method still outperforms the state of the art on MNIST, and performs on pair with CQ on LabelMe22K

Very large-scale training/query/base datasets. Finally, we report results on two very large-scale datasets: SIFT10M and SIFT1B in Figure 4, with detailed results in Table 3. Interestingly, the performance gap between our method and our baselines is more pronounced in these datasets, suggesting that the performance advantage of LSQ increases for larger datasets (as opposed to OPQ, whose gap over PQ consistently shrinks when more data is available). On SIFT1B with 64 bits, LSQ-32 shows a relative improvement of 13% in recall@1 over CQ, our closest competitor, and consistently obtains between 2 and 3 points of advantage in recall over CQ in other cases. These are, to the best of our knowledge, the best results reported on SIFT1B using exhaustive search so far.

Sparse extension. Following Zhang *et al.* [12], we evaluated the sparse version of our method using 2 different levels of sparsity: SLSQ1, with $\|C\|_0 \leq S = h \cdot d$, which has a query time comparable to PQ, and SLSQ2, with $\|C\|_0 \leq S = h \cdot d + d^2$, which

	SIFT10M – 64 bits			SIFT10M – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	15.79	50.86	86.57	39.31	54.74	74.89
OPQ	17.49	54.92	89.41	40.80	56.73	77.15
CQ [11]	21	63	93	47	64	84
LSQ-16	22.51	64.62	94.67	49.26	66.75	85.36
LSQ-32	**22.94**	**65.20**	**94.85**	**49.50**	**67.31**	**86.33**

	SIFT1B – 64 bits			SIFT1B – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	06.34	24.41	56.92	26.38	38.57	56.45
OPQ	07.02	27.34	61.89	28.43	40.80	59.53
CQ [11]	09	33	70	34	48	68
LSQ-16	09.73	35.82	73.84	35.32	50.84	70.66
LSQ-32	**10.18**	**36.96**	**75.31**	**36.35**	**51.99**	**72.13**

Table 3. Detailed recall@N values for our method on large-scale datasets: SIFT10M and SIFT1B.

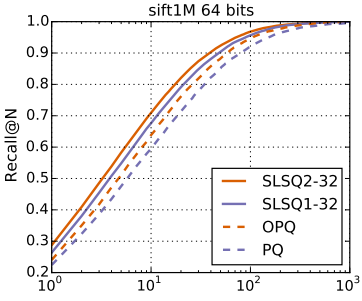

Fig. 5. Recall@N curves for our sparse methods SLSQ1 and SLSQ2 on SIFT1M.

	SIFT1M – 64 bits		
	R@1	R@10	R@100
PQ	22.53 ± 0.31	60.14 ± 0.41	91.99 ± 0.17
SCQ1 [12]	25	67	95
SLSQ1-16	25.88 ± 0.31	66.69 ± 0.38	95.26 ± 0.14
SLSQ1-32	**26.36** ± 0.20	**67.61** ± 0.08	**95.66** ± 0.07
OPQ	24.34 ± 0.52	63.89 ± 0.30	94.04 ± 0.08
SCQ2 [12]	27	68	96
SLSQ2-16	28.04 ± 0.40	70.10 ± 0.25	96.56 ± 0.08
SLSQ2-32	**28.72** ± 0.16	**70.94** ± 0.32	**96.81** ± 0.08

Table 4. Recall@N values for the sparse extension of our method on SIFT1M using 64 bits.

has a query time comparable to OPQ. We compare against SCQ1 and SCQ2 from [12], which have the same levels of sparsity. We focus on SIFT1M; as noted in Section 4, substantial speedups can be obtained on small datasets by using sparse codebooks.

Again, in this scenario we observed improved performance compared to our baselines. While when using dense codebooks our method achieved a small gain of 0.79 in recall@1 over CQ, in this case the improvement jumps to 1.36 and 1.72 over SCQ. This is comparable to the 1.80 gain that OPQ achieves over PQ, and virtually equalizes the performance of CQ at 29: compared to CQ, our SLSQ2-32 method only loses 0.28 points in recall when using sparse codebooks (and thus having lower query time).

Encoding speed and comparison to CQ. In Table 5 we show the speed advantage that sharing the pairwise terms gives to LSQ over a naïve implementation that encodes each point sequentially. The table shows that our method, implemented in a high-level language, remains fast when using up to 32 ILS iterations, handily achieving speeds faster than real-time (we believe that even better performance could be achieved with a C implementation). We also implemented a version of our method using an Nvidia GTX Titan X GPU. It took a novice CUDA developer about 2 days to complete this implementation, which again highlights the simplicity of our method (and suggests that better speeds are possible). Using this implementation, it is possible to encode SIFT1B using 128 bits and 32 ILS iterations in around 1.5 days.

Looking at our results, it is clear that our main competitor is CQ [11]. Our method has demonstrated higher recall on all datasets and benefits from a simpler formulation.

Method	Sequential		Batched	
codebooks (m)	7	15	7	15
LSQ-16 (ms.)	1.52	7.02	0.53	2.01
LSQ-32 (ms.)	3.01	13.93	1.05	4.03

Method	GPU (batched)	
codebooks (m)	7	15
LSQ-16 (μs)	17.9	67.2
LSQ-32 (μs)	35.7	134.3

Method	Exhaustive NN	
codebooks (m)	8	16
PQ (μs)	42.6	77.9
OPQ (μs)	49.2	90.3

Table 5. Time spent per vector during encoding in our approach. "Sequential" refers to an LSQ implementation where ICM encodes each point sequentially (*i.e.*, does not take advantage of the shared pairwise terms). "Batched" is our LSQ implementation, which performs conditioning of shared pairwise terms among several data points.

Being free of additional constraints, our method is also better suited to make use of state-of-the-art L1 optimizers in the sparse codebook case, where we have also demonstrated state-of-the-art performance. Perhaps the most obvious downside of our method is encoding time. While CQ mentions using 3 ICM iterations, our method uses either 16, 64 or 128 iterations. In query/base partitions, we use 16 ICM iterations in total, which is not too large an overhead over the 3 iterations of CQ. Importantly, this protocol has been suggested as the most suitable for very large-scale datasets with inverted indices [27, 34]. Regarding training/query/base partitions, the 64 and 128 ICM iterations of our method may appear to be a large overhead over CQ. However, we note that unlike CQ, our method does not require dataset-specific hyperparameter optimization. The authors of CQ have optimized the penalty parameter of L-BFGS for recall [11, 12], which means the the method is actually run several times to find the best parameter value. In contrast, our method uses the same parameter settings for all datasets and only focuses on minimizing quantization error. CQ tries out ~ 10 different values of its hyperparameter[6], and thus ICM is run ~ 30 times. In that case, our method has only a $2-4\times$ overhead over CQ in training/query/base partitions, and is overall faster in query/base datasets. In any case, in a practical application one may always resort to our GPU implementation to offset the one-time cost of database encoding.

7 Conclusion

We have introduced a new method for solving the encoding problem in AQ based on stochastic local search (SLS). The high encoding performance of our method demonstrates that the elegant formulation introduced by AQ can be leveraged to achieve an improvement over the current state of the art in multi-codebook quantization. We have also shown that our method can be easily extended to accommodate sparsity constraints in the codebooks, which results in another conceptually simple method that also outperforms its competitors.

Acknowledgements. We thank NVIDIA for the donation of some of the GPUs used in this project. Joris Clement was supported by DAAD while doing an internship at the University of British Columbia. This research was supported in part by NSERC.

[6] Personal communication.

References

1. Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: exploring photo collections in 3d. TOG **25**(3) (2006) 1
2. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. IJCV **60**(2) (2004) 1
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR. (2009) 1
4. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: A large data set for non-parametric object and scene recognition. TPAMI **30**(11) (2008) 1
5. Gong, Y., Lazebnik, S.: Iterative quantization: A Procrustean approach to learning binary codes. In: CVPR. (2011) 1
6. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS. (2009) 1
7. Babenko, A., Lempitsky, V.: Additive quantization for extreme vector compression. In: CVPR. (2014) 1, 2, 3, 4, 5, 6, 8, 9, 10
8. Babenko, A., Lempitsky, V.: Tree quantization for large-scale similarity search and classification. In: CVPR. (2015) 1, 2, 3, 5, 6, 8
9. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization. TPAMI **36**(4) (2014) 1, 2, 8, 9
10. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. TPAMI **33**(1) (2011) 1, 2, 8, 9
11. Zhang, T., Du, C., Wang, J.: Composite quantization for approximate nearest neighbor search. In: ICML. (2014) 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13, 14
12. Zhang, T., Qi, G.J., Tang, J., Wang, J.: Sparse composite quantization. In: CVPR. (2015) 1, 6, 7, 8, 9, 12, 13, 14
13. Guo, R., Kumar, S., Choromanski, K., Simcha, D.: Quantization based fast inner product search. In: AISTATS. (2016) 1
14. Shrivastava, A., Li, P.: Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In: NIPS. (2014) 1
15. Jiaxiang Wu, Cong Leng, Y.W.Q.H., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: CVPR. (2016) 1
16. Järvisalo, M., Le Berre, D., Roussel, O., Simon, L.: The international SAT solver competitions. AI Magazine **33**(1) (2012) 89–92 2
17. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations & applications. Elsevier (2004) 2, 4, 6, 11
18. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research (2008) 565–606 2
19. Norouzi, M., Fleet, D.J.: Cartesian k-means. In: CVPR. (2013) 2, 9
20. Gersho, A., Gray, R.M.: Vector quantization and signal compression. Kluwer Academic Publishers (1992) 2, 3
21. Cooper, G.F.: The computational complexity of probabilistic inference using Bayesian belief networks. AI **42**(2) (1990) 3
22. Kappes, J.H., Andres, B., Hamprecht, F., Schnorr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B.X., Lellmann, J., Komodakis, N., et al.: A comparative study of modern inference techniques for discrete energy minimization problems. In: CVPR. (2013) 1328–1335 3, 5
23. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. Informs Journal on Computing **25**(2) (2013) 346–363 4
24. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: VISAPP (1). (2009) 6
25. Babenko, A., Lempitsky, V.: The inverted multi-index. In: CVPR. (2012) 6
26. Xia, Y., He, K., Wen, F., Sun, J.: Joint inverted indexing. In: ICCV. (2013) 6

27. Babenko, A., Lempitsky, V.: Improving bilayer product quantization for billion-scale approximate nearest neighbors in high dimensions. arXiv preprint arXiv:1404.1831 (2014) 6, 10, 14
28. André, F., Kermarrec, A.M., Le Scouarnec, N.: Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan. Proceedings of the VLDB Endowment **9**(4) (2015) 288–299 6
29. Van Den Berg, E., Friedlander, M.P.: Probing the pareto frontier for basis pursuit solutions. SIAM Journal on Scientific Computing **31**(2) (2008) 890–912 7
30. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: Delving deep into convolutional nets. In: BMVC. (2014) 8
31. Babenko, A., Slesarev, A., Chigorin, A., Lempitsky, V.: Neural codes for image retrieval. ECCV (2014) 8
32. Razavian, A.S., Azizpour, H., Sullivan, J., Carlsson, S.: CNN features off-the-shelf: an astounding baseline for recognition. In: Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on, IEEE (2014) 512–519 8
33. Norouzi, M., Fleet, D.J.: Minimal loss hashing for compact binary codes. In: ICML. (2011) 9
34. Kalantidis, Y., Avrithis, Y.: Locally optimized product quantization for approximate nearest neighbor search. In: CVPR. (2014) 9, 14
35. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. arXiv preprint arXiv:1411.1607 (2014) 10