Stacked Quantizers for Compositional Vector Compression

Julieta Martinez Holger H. Hoos James J. Little
University of British Columbia
{julm,hoos,little}@cs.ubc.ca

Abstract

Recently, Babenko and Lempitsky [3] introduced Additive Quantization (AQ), a generalization of Product Quantization (PQ) where a non-independent set of codebooks is used to compress vectors into small binary codes. Unfortunately, under this scheme encoding cannot be done independently in each codebook, and optimal encoding is an NP-hard problem. In this paper, we observe that PQ and AQ are both compositional quantizers that lie on the extremes of the codebook dependence-independence assumption, and explore an intermediate approach that exploits a hierarchical structure in the codebooks. This results in a method that achieves quantization error on par with or lower than AQ, while being several orders of magnitude faster. We perform a complexity analysis of PQ, AQ and our method, and evaluate our approach on standard benchmarks of SIFT and GIST descriptors, as well as on new datasets of features obtained from state-of-the-art convolutional neural networks.

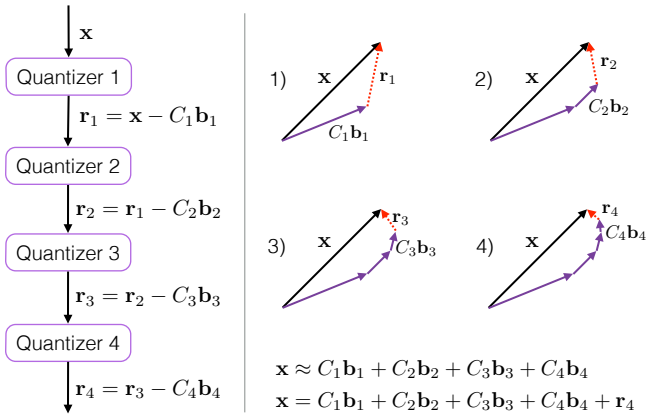

Figure 1. Encoding as performed with Stacked Quantizers, shown for 4 subcodebooks. Left: the vector is passed through a series of quantizers, with residuals further encoded down the line. Right: A geometric interpretation of our approach. After recursively encoding residuals, the representation is additive in the encodings, and the quantization error is the remaining residual.

1. Introduction

Vector quantization has established itself as a default approach to scale applications such as visual recognition and image retrieval. Quantization is usually performed on large datasets of local descriptors (*e.g.*, SIFT [13]), or global representations (*e.g.*, VLAD [12] or Fisher vectors [17]). Recent work has also explored the performance-vs.-compression trade-off in state-of-the-art features obtained from deep convolutional neural networks [6]. Outside the computer vision community, vector quantization is also studied in information theory, multimedia retrieval and unsupervised learning.

Vector quantization is usually posed as the search for a set of *codewords* (*i.e.*, a *codebook*) that minimize quantization error. The problem can be solved in a straightforward manner with the k-means algorithm which, unfortunately, scales poorly for large codebooks. While larger codebooks achieve lower quantization error, the downside is that encoding and search times scale linearly with codebook size.

Several algorithms, such as kd-trees and hierarchical k-means, alleviate the search and encoding problems by indexing the codebook with complex data structures [14], achieving sublinear search time as a trade-off for recall. These approaches, however, have a large memory footprint, since all the uncompressed vectors must be kept in memory.

Another line of research considers approaches with an emphasis on low memory usage, compressing vectors into small binary codes. While for a long time hashing approaches were the dominant trend [10, 19], they were shown to be largely outperformed by Product Quantization (PQ) [11]. PQ is a compositional vector compression algorithm that decomposes the data into orthogonal subspaces and quantizes each subspace independently. As a result, vectors can be encoded independently in each subspace, and distances between uncompressed queries and the database can be efficiently computed through a series of table lookups. This combination of small memory footprint, low quantization error and fast search makes PQ a very at-

tractive approach for scaling computer vision applications.

Recently, Babenko and Lempitsky [3] introduced Additive Quantization (AQ), a generalization of PQ that retains its compositional nature, but is able to handle subcodebooks of the same dimensionality as the input vectors. With a few caveats, AQ can also be used for fast approximate nearest neighbour search and consistently achieves lower quantization error than PQ. However, since the codebooks are no longer pairwise orthogonal (*i.e.*, no longer independent), encoding cannot be done independently in each subspace. In [3], beam search was proposed as a solution to this problem, but this results in very slow encoding, which greatly limits the scalability of the proposed solution.

In this paper, we first analyze PQ and AQ as compositional quantizers, under a framework that makes the simplifying assumptions of PQ w.r.t. AQ rather evident. We next investigate the computational complexity implications resulting from the differences between AQ and PQ, and finally derive an intermediate approach that retains the expressive power of AQ, while being only slightly slower than PQ.

Our approach compares favourably to AQ in 3 ways: (i) it consistently achieves similar or lower quantization error (and therefore, lower error than PQ), (ii) it is *several orders of magnitude faster* and (iii), it is also simpler to implement.

2. Background and related work

We introduce some notation mostly following [15]. We review the vector quantization problem, the scalability approaches proposed by PQ and AQ, and discuss their advantages and disadvantages.

2.1. Vector quantization

Given a set of vectors $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, the objective of vector quantization is to minimize the quantization error, *i.e.*, to determine

$$\min_{C,\mathbf{b}} \frac{1}{n} \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - C\mathbf{b}\|_2^2, \tag{1}$$

where $C \in \mathbb{R}^{d \times k}$ contains k cluster centers, and $\mathbf{b} \in \{0,1\}^k$ is subject to the constraints $\|\mathbf{b}\|_0 = 1$ and $\|\mathbf{b}\|_1 = 1$. That is, $\mathbf{b}$ may only index into one entry of C. C is usually referred to as a *codebook*, and $\mathbf{b}$ is called a *code*.

If we let $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ contain all the $\mathbf{x} \in \mathcal{X}$, and similarly let $B = [\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n] \in \{0,1\}^{k \times n}$ contain all the codes, the problem can be expressed more succinctly as determining

$$\min_{C,B} \frac{1}{n} \|X - CB\|_2^2. \tag{2}$$

Without further constraints, one may solve expression 2 using the k-means algorithm, which alternatively solves for B (typically exhaustively computing the distance to the k clusters in C for each point in X) and C (finding the mean of each cluster) until convergence. The performance of k-means is better as the size of the codebook, k, grows larger but, unfortunately, the algorithm is infeasible for large codebook sizes (for example, $k = 2^{64}$ clusters would far exceed the memory capacity of current machines). The challenge is thus to handle large codebooks that achieve low quantization error while having low memory overhead.

2.2. Compositional quantization models

One way of scaling the codebook size looks at compositional models, where smaller subcodebooks can be combined in different ways to potentially represent an exponential number of clusters. Compositional quantization can be formulated similarly to k-means, but restricted to a series of constraints that introduce interesting computational tradeoffs. The objective function of compositional quantization can be expressed as

$$\min_{C_i,\mathbf{b}_i} \frac{1}{n} \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - \sum_{i}^{m} C_i \mathbf{b}_i\|_2^2, \tag{3}$$

that is, the vector $\mathbf{x}$ can be approximated not only by a single codeword indexed by its code $\mathbf{b}$, but by the *addition* of its encodings in a series of codebooks. We refer to the C_i as *subcodebooks*, and similarly call the $\mathbf{b}_i$ *subcodes*. We let each subcodebook contain h cluster centres: $C_i \in \mathbb{R}^{d \times h}$, and each subcode $\mathbf{b}_i$ remains limited to having only one non-zero entry: $\|\mathbf{b}_i\|_0 = 1$, $\|\mathbf{b}_i\|_1 = 1$. Since each $\mathbf{b}_i$ may take a value in the range$[1, 2, \dots, h]$, and there are m subcodes, the resulting number of possible cluster combinations is equal to h^m, *i.e.*, superlinear in m. Now we can more succinctly write expression 3 as

$$\|X - CB\|_2^2 = \|X - [C_1, C_2, \dots, C_m] \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{bmatrix} \|_2^2, \tag{4}$$

where $B_i = [\mathbf{b}_{i1}, \mathbf{b}_{i2}, \dots, \mathbf{b}_{in}] \in \{0,1\}^{h \times n}$. As we will show next, AQ, PQ and Optimized Product Quantization (OPQ) [15, 9] belong to this family of models.

2.2.1 Product Quantization

PQ can be formulated right away with Eq. 4 under the constraint that all the subcodebooks be pairwise orthogonal [15]:

$$\forall i, j : i \neq j \rightarrow C_i^\top C_j = 0_{h \times h}, \tag{5}$$

that is, C is blockwise diagonal [15]:

$$C = [C_1, C_2, \ldots, C_m] = \begin{bmatrix} D_1 & 0 & \ldots & 0 \\ 0 & D_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & D_m \end{bmatrix}, \quad (6)$$

where the entries $D_i \in \mathbb{R}^{(d/m)\times h}$ are the only non-zero components of C. This constraint assumes that the data in X was generated from a series of mutually independent subspaces (those spanned by the subcodebooks C_i), which is rarely the case in practice. There are, however, some advantages to this formulation.

The subcodebook independence of PQ offers 3 main advantages,

1. Under the orthogonality constraint we can efficiently learn the subcodebooks $\mathcal{C}_i$ by independently running k-means on d/m dimensions. The complexity of k-means is $\mathcal{O}(nkdi)$ for n datapoints, k cluster centres, d dimensions and i iterations. PQ solves m d/m-dimensional k-means problems with h cluster centres each, resulting in a complexity of $\mathcal{O}(mnh(d/m)i) = \mathcal{O}(nhdi)$; *i.e.*, training PQ is as complex as solving a k-means problem with h cluster centres.

2. Once training is done, the encoding of the database can also be performed efficiently in $\mathcal{O}(nhd)$ (in line with k-means), which is essential for very large databases.

3. Distance computation between a query $\mathbf{q}$ and a encoded vector $\sum_{i=1}^{m} C_i\mathbf{b}_i$ is efficient because the subcodebooks are orthogonal, and therefore the total distance is equal to the sum of the distances in each subspace [11]: $\|\mathbf{q} - \sum_{i=1}^{m} C_i\mathbf{b}_i\|_2^2 = \sum_{i=1}^{m}\|\mathbf{q}_i - D_i\mathbf{b}_i\|_2^2$, where $\mathbf{q} = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m]$, and $\mathbf{q}_i \in \mathbb{R}^{d/m}$. These distances can be precomputed for each query and quickly evaluated with m table lookups. This is called Asymmetric Distance Computation in [11] and is the mechanism that makes PQ attractive for fast approximate nearest neighbour search.

2.2.2 Optimized Product Quantization

One of the main disadvantages of PQ is that X is forced to fit in a model that assumes that the data was generated from statistically independent subspaces. Lower quantization error can be achieved if more degrees of freedom are added to the model. In particular, since rotation is a distance-preserving operation, it seems natural to experiment with codebook rotations that minimize quantization error. In OPQ, the objective function becomes [15]

$$\min_{R,C,B} \frac{1}{n}\|X - RCB\|_2^2, \quad (7)$$

where C and B are expanded as in Eq. 4, and R belongs to the Special Orthogonal Group $SO(d)$. In this sense, PQ is a special case of OPQ where R is the d-dimensional identity matrix: $R = I_d$. Independently, Ge *et al*. [9] and Norouzi & Fleet [15] proposed an iterative method similar to Iterative Quantization [10] that optimizes R in expression 7. Notice, however, that the orthogonality constraint is maintained from PQ to OPQ.

Lower quantization error can be achieved if the independence assumption is not enforced, at the cost of more complex encoding and distance computation. These trade-offs were first introduced in [3] and called Additive Quantization (AQ). We briefly review AQ here.

2.2.3 Additive Quantization

In AQ, the subspaces spanned by the subcodebooks C_i are not mutually orthogonal (*i.e.*, not mutually independent). Formally, and although not explicitly stated in [3], AQ solves the formulation of Eq. 3 without any further constraints. This makes of AQ a strictly more general model than PQ/OPQ. However, this complexity comes at a cost.

The subcodebook dependence of AQ comes with 3 main disadvantages with respect to PQ/OPQ,

1. The distance between a query $\mathbf{q}$ and a encoded vector $\sum_{i=1}^{m} C_i\mathbf{b}_i$ cannot be computed with m table lookups. However, it can be found using the identity

$$\|\mathbf{q} - \sum_{i=1}^{m} C_i\mathbf{b}_i\|_2^2 = \\ \|\mathbf{q}\|_2^2 - \sum_{i=1}^{m} 2\langle \mathbf{q}, C_i\mathbf{b}_i\rangle + \|\sum_{i=1}^{m} C_i\mathbf{b}_i\|_2^2 \quad (8)$$

   where the first term is a constant and does not affect the query ranking; the second term can be precomputed and stored for fast evaluation with m table lookups, and the third term can either be precomputed and quantized for each vector in the database (at an additional memory cost), or can be computed on the fly as

$$\|\sum_{i=1}^{m} C_i\mathbf{b}_i\|_2^2 = \sum_{i}^{m}\|C_i\mathbf{b}_i\|_2^2 + 2\sum_{i\neq j}^{m}\langle C_i\mathbf{b}_i, C_j\mathbf{b}_j\rangle \quad (9)$$

   where the terms can also be precomputed and retrieved in m table lookups. Thus, AQ has either a time ($2m$

vs. m lookups) or memory overhead (for storing the quantized result of Eq. 9) during distance computation with respect to PQ. Although this may sound as a major problem for AQ, it was shown in [3] that sometimes the distortion error gain can be high enough that allocating memory from the code budget to store the result of Eq. 9 results in better recall *and* faster distance computation compared to PQ/OPQ. This motivates us to look for better solutions to the AQ formulation.

2. For a given set of subcodebooks C_i and a vector $\mathbf{x}$, encoding amounts to choosing the optimal set of codes $\mathbf{b}_i$ that minimize quantization error $\|\mathbf{x} - \sum_{i=1}^{m} C_i \mathbf{b}_i\|_2^2$. Unfortunately, without the orthogonality constraint the choice of $\mathbf{b}_i$ cannot be made independently in each subcodebook. This means that, in order to guarantee optimality, the search for the best encoding must be done over a combinatorial space of codewords. Moreover, it was shown in [3] that this problem is equivalent to inference on a fully connected pairwise Markov Random Field, which is well-known to be NP-hard [7].

   Since brute force search is not possible, one must settle for a heuristic search method. Beam search was proposed as a solution in [3], resulting in rather slow encoding. Beam search is done in m iterations. At iteration i the distance is computed from each of the b candidate solutions to the set of $k \cdot (m - i)$ plausible candidates (in the $m - i$ codebooks that have not contributed to the candidate solution). At the end of the iteration we have b^2 candidate solutions, from which the top b are kept as seeds for the next iteration [3]. The complexity of this process is $\mathcal{O}(m^2 mbhd) = \mathcal{O}(m^3 bhd)$, where b is the search depth. As we will show, this makes the original solution of AQ impractical for very large databases.

3. Training consists of learning the subcodebooks C_i and subcodebook assignments $\mathbf{b}_i$ that minimize expression 3. A typical approach is to use coordinate descent by fixing the subcodebooks C_i while updating the codes $\mathbf{b}_i$ (encoding), and later fixing $\mathbf{b}_i$ while updating C_i (codebook update). As a side effect of slow encoding, we find that training is also very slow in AQ. While this might seem as a minor weakness of AQ (since training is usually done off-line, without tight time constraints), having faster training also means that for a fixed time budget we can handle larger amounts of training data. In the quantization setting, this means that we can use a larger sample to better capture the underlying distribution of the database.

   In [3], codebook update is done by solving the over-constrained least-squares problem that arises from Eq. 4 when holding B fixed and solving for C. Fortunately, this decomposes into d independent subproblems of n equations over mh variables [3]. This corresponds to an optimal codebook update in the least squares sense. We find that compared to encoding this step is rather fast, and thus focus on speeding up encoding.

3. Stacked Vector Quantizers

Within the subcodebook dependence-independence framework introduced in Section 2, we can see that PQ and OPQ assume subcodebook independence, while AQ embraces the dependence and tries to solve a more complex problem. As we will show next, there is a fertile middle ground between these approaches. We propose a *hierarchical* assumption, which has the advantage of being fast to solve while maintaining the expressive power of AQ. We now introduce our proposed approach to compositional quantization.

Stacking Quantizers. Due to the superior performance of AQ, we want to maintain its key property: subcodebook dependence. However, we look for a representation that can compete with PQ in terms of fast training and good scalability, for which fast encoding is essential. We propose to use a hierarchy of quantizers (see Figure 1, left), where the vector is sequentially compressed in a coarse-to-fine manner.

Encoding. Fast encoding is at the heart of our approach. We assume that the subcodebooks C_i have a hierarchical structure, where C_1 gives the coarsest quantization and C_M the finest. Encoding is done greedily. In the first step, we choose the code $\mathbf{b}_1$ that most minimizes the quantization error $\|\mathbf{x} - C_1 \mathbf{b}_1\|_2^2$. Since all the subcodebooks are small, the search for $\mathbf{b}_1$ can be done exhaustively (as in k-means).

Next, we compute the first residual $\mathbf{r}_1 = \mathbf{x} - C_1 \mathbf{b}_1$. We now quantize $\mathbf{r}_1$ using the codewords in C_2, choosing the one that minimizes the quantization error $\|\mathbf{r}_1 - C_2 \mathbf{b}_2\|_2^2$. This process is repeated until we run out of codebooks to quantize residuals, with the last residual $\mathbf{r}_m$ being equal to the total quantization error (see Figure 1, right). Now it is clear that we satisfy our first desired property, as the representation is additive in the encodings: $\mathbf{x} \approx \sum_{i=1}^{m} C_i \mathbf{b}_i$, and the codewords all are d-dimensional (*i.e.*, not independent of each other).

The complexity of this step is $\mathcal{O}(mhd)$ for m subcodebooks, each having h subcodewords, and a vector of dimensionality d. This corresponds to a slight increase in computation with respect to PQ ($\mathcal{O}(hd)$), but is much faster than AQ ($\mathcal{O}(m^3 bhd)$). Given that encoding is only slightly more expensive than PQ, we can say that we have also achieved our second desired property.

Initialization. The goal of initialization is to create a coarse-to-fine set of codebooks. This can be achieved by simply performing k-means on X, obtaining residuals by subtracting the assigned codewords, and then performing k-means on the residuals until we run out of codebooks.

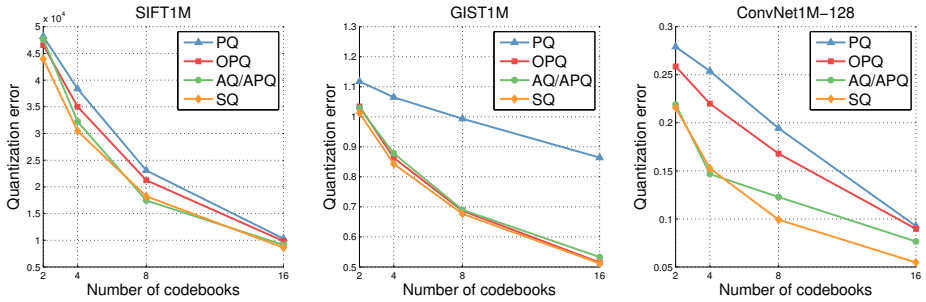

Figure 2. Quantization error on SIFT1M, GIST1M and ConvNet1M 128. SQ shows the biggest performance advantage on deep features.

Formally, in the first step we obtain C_1 from the cluster centres computed by k-means on X, and we obtain residuals by subtracting $R_1 = X - C_1 B_1$. In the second step we obtain C_2 from k-means on R_1, and the residuals are refined to $R_2 = R_1 - C_2 B_2$. This process continues until we run out of codebooks (notice how this both is analogous to, and naturally gives rise to, the fast encoding proposed before). By the end of this initialization, we have an initial set of codebooks $C = [C_1, C_2, \ldots, C_m]$ that have a hierarchical structure, and with which encoding can be performed in a greedy manner.

The computational cost of this step is that of running k-means on n vectors m times, *i.e.*, $\mathcal{O}(mnhdi)$ for subcodebooks of size h, dimensionality d and i k-means iterations.

Codebook refinement. The initial set of codebooks can be further optimized with coordinate descent. This step is based on the observation that, during initialization, we assume that in order to learn codebook C_i we only need to know codebooks $C_1, C_2, \ldots, C_{i-1}$. However, after initialization all the codebooks are fixed. This allows us to fine-tune each codebook given the value of the rest.

Although it is tempting to use the least-squares-optimal codebook update proposed in [3], we have found that this tends to destroy the hierarchical subcodebook structure resulting from initialization. Without a hierarchical structure encoding cannot be done fast, which is one of the key properties that we wish to maintain. We therefore propose an ad hoc codebook refinement technique that preserves the hierarchical structure in the codebooks.

Let us define $\hat{X}$ as the approximation of X from its encoding

$$\hat{X} = CB. \tag{10}$$

Now, let us define $\hat{X}^{-i}$ as an approximation to the original dataset X obtained using the learned codebooks $[C_1, C_2, \ldots, C_m]$ and codes $B = [B_1^\top, B_2^\top, \ldots, B_n^\top]^\top$, *except* for C_i, *i.e.*,

$$\hat{X}^{-i} = \hat{X} - C_i B_i. \tag{11}$$

We can now see that the optimal value of C_i given the rest of the codebooksis obtained by running k-means on $X - X^{i-1}$, *i.e.*, the residual after removing the contribution of the rest of the codebooks. Since we already know the cluster membership to C_i (*i.e.*, we know B_i) either from initialization or the previous iteration, we need to update only the cluster centres instead of restarting k-means (similar to how OPQ updates the codebooks given an updated rotation [9, 15]).

Enforcing codebook hierarchy is of the essence. Therefore, we run our codebook update in a top-down manner. We first update C_1 and update all codes. Next, we update C_2 and update codes again. We repeat the process until we have updated C_m, followed by a final update of the codes. Updating the codes after each codebook update ensures that the codebook hierarchy is maintained. A round of updates from codebooks 1 to m amounts to one iteration of our codebook refinement.

The algorithm involves encoding using m codebooks in the first pass, $m - 1$ in the second pass, $m - 2$ in the third pass and so on until only one set of codes is updated. This means that the time complexity of the codebook refinement procedure is quadratic in the number of codebooks. This is a significant increase with respect to PQ/OPQ, which are linear in m during their training, but also represents an important reduction against the cubic scaling of AQ. Also, notice that the training usually has to be done only once with a small data sample, and database encoding remains efficient.

4. Experiments

Our main interest is to reduce quantization error because it has been demonstrated to lead to better retrieval recall, mean average precision and classification performance [3, 9, 11, 15]. We also demonstrate two applications of our method: (i) approximate nearest neighbour search and (ii) classification performance with compressed features. In all our experiments we use codebooks of size $h = 256$; this means that 2, 4, 8 and 16 codebooks generate codes of 16, 32, 64 and 128 bits.

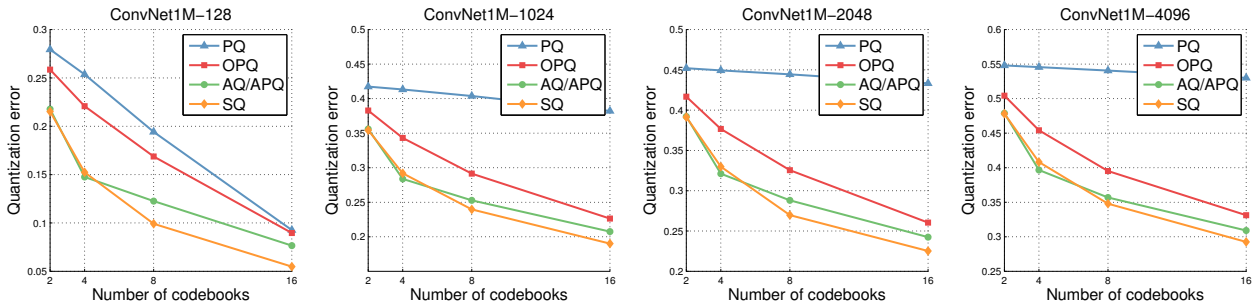

Figure 3. Quantization error on deep features with different dimensionalities. The non-independent approaches AQ and SQ clearly outperform PQ and OPQ. SQ achieves the best performance when using 8 and 16 codebooks (64 and 128 bits per feature) in all cases.

Datasets. We test our method on three datasets. The first two are SIFT1M and GIST1M, introduced in [11]. SIFT1M consists of 128-dimensional SIFT [13] descriptors, and GIST1M consists of 960-dimensional GIST [16] descriptors. Since hand-crafted features are consistently being replaced by features obtained from deep convolutional neural networks, we also consider a dataset of deep features: ConvNet1M-128. We obtained ConvNet1M-128 by computing 128-dimensional deep learning features on the ILSVRC-2012 training dataset [8] using the CNN-M-128 network provided by Chatfield *et al*. [5] and then subsampling equally at random from all classes. This network follows the architecture proposed by Zeiler and Fergus [20], with the exception that the last fully-connected layer was reduced from 4096 to 128 units. It has been shown that this intra-net compression has a minimal effect on classification performance [5], and exhibits state-of-the-art accuracy on image retrieval [6]. However, to the best of our knowledge we are the first to benchmark quantization techniques on deep learning features. We obtained the features from a central 224×224 image crop without further data augmentation. In the three datasets 100 000 vectors are given for training, 10 000 for query and 1 000 000 for database.

Baselines. We compare against 3 baselines. The first one is AQ as proposed by Babenko and Lempitsky [3], which consists of beam search for encoding and a least-squares codebook update in an iterative manner. As in [3], we set the beam search depth b to 16 during training and to 64 for the database encoding. Although [3] does not mention the number of iterations used during training, we found that 10 iterations reproduce the results reported by the authors and, as we will show, this is already several orders of magnitude slower than our approach. Since encoding scales cubically with the number of codebooks, for code lengths of 64 and 128 bits (8 and 16 codebooks respectively) we use the hybrid APQ algorithm suggested in [3], where the dataset is first preprocessed with OPQ, and then groups of 4 subcodebooks are refined independently with AQ. APQ was proposed for practical reasons, as otherwise AQ would require several days to complete given more than 4 subcodebooks: the need for this approximation starts to show the poor scalability of AQ. Since no code for AQ is available, we wrote our own implementation and incorporated the optimizations suggested in [3]. We will make all our code available, including this baseline.

The second baseline is Optimized Product Quantization [9, 15], which was briefly introduced in Section 2. We use the publicly available implementation by Norouzi & Fleet[1], and set the number of optimization iterations to 100. The third baseline is Product Quantization [11]. We slightly modified the OPQ code to create this baseline. We also use 100 iterations in PQ.

4.1. Quantization Error

Our main quantization results are shown on Figure 2. First, we observe that our method has a performance similar to AQ on SIFT1M and GIST1M. This is already good news, given the better scalability of our method. Moreover, we note that SQ obtains a large advantage on the deep features of ConvNet1M-128 when using 8 and 16 codebooks. We find this result rather encouraging, as deep features are likely to replace hand-crafted descriptors such as SIFT and GIST in the foreseeable future.

OPQ achieves a large gain compared to PQ in GIST1M, and this gap is only slightly improved by AQ and SQ. Since both SIFT1M and ConvNet1M-128 have low dimensionality (128), and GIST1M has high-dimensional descriptors (960), it remains unclear whether the advantages of AQ and SQ are only restricted to low-dimensional descriptors. We investigate this question by benchmarking the methods on 1024-, 2048- and 4096-dimensional deep features obtained in a similar manner to ConvNet1M-128, but using using the CNN-M-1024, CNN-M-2048 and CNN-M networks from [5] respectively. The quantization results on these datasets are shown on Figure 3. While the PQ-to-OPQ gap is still present for high-dimensional features, we see that AQ and SQ maintain a performance gap from OPQ

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

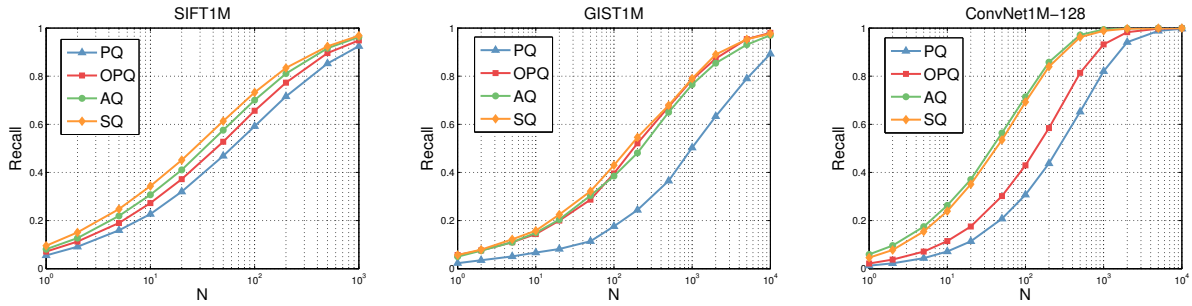

Figure 4. Recall@N on the SIFT1M, GIST1M and Convnet1M-128 datasets. Because of limited space, we consider only 32-bit codes (4 codebooks). We confirm previous observations [3, 15, 9] that correlate quantization error with nearest neighbour search performance: our method has the best recall for all values of N in the SIFT1M and GIST1M datasets. SQ is slightly outperformed by AQ on ConvNet1M-128, but still performs much better than PQ and OPQ. We show results for longer codes in the supplementary material.

similar to that observed on the 128-dimensional features. Moreover, our method remains the clear winner for 8 and 16 codebooks, and largely competitive with AQ for 4 codebooks. These results suggest that codebook independence hurts the compression of deep features particularly badly and motivates more research of compositional quantization methods that follow the formulation of expression 3.

4.2. Approximate Nearest Neighbour Search

We demonstrate the performance of our method on fast search of K nearest neighbours with recall@N curves [11]. These curves represent the probability of the true K nearest neighbours being in a retrieved list of N neighbours for varying N. We set $K = 1$ and observe little variability for other values. Our main results are shown on Figure 4. As expected, lower quantization error lets us achieve higher recall on SIFT1M and GIST1M, although on GIST1M OPQ and AQ achieve very competitive performance. On ConvNet1M-128, our method was slightly outperformed by AQ; however, this trend is reversed for longer codes, consistent with the quantization error of Fig. 2. We show results on longer codes in the supplementary material.

4.3. Large-Scale Object Classification

We study the trade-off in classification performance vs. compression rate on the ILSVRC-2012 dataset using deep learning features. We trained a linear SVM on the 1.2 million uncompressed examples provided, and preprocessed the features with L2 normalization, which was found to improve performance in [5]. The 50 000 images in the validation set were preprocessed similarly and compressed before evaluation. This scenario is particularly useful when one wants to search for objects in large unlabelled datasets [1, 4], and in retrieval scenarios where classifiers are applied to large collections of images in search for high scores [6, 18]. Notice that in this scenario, the only operation needed between the support vectors and the database descriptors is a dot product; as opposed to distance computation, this can be done with m lookups in AQ and SQ, the same as for PQ and OPQ. We report the classification error taking into account the top 5 predictions.

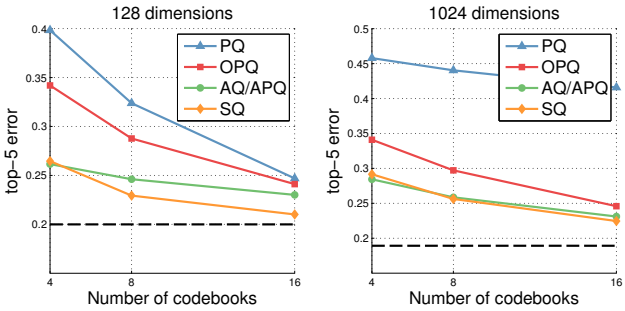

Figure 5. Top-5 classification error on the ILSVRC-2012 dataset as a function of compression. The dotted black line corresponds to performance without compression. The left pane shows performance using 128-dimensional deep features, and the right pane shows performance for 1024-dimensional deep features.

Classification results are shown on Figure 5. We observe a similar trend to that seen in our quantization results, with PQ and OPQ consistently outperformed by AQ and SQ. Using 128-dimensional features our method performs similarly to AQ using 4 codebooks, but shows better performance for larger code sizes. Using 1024-dimensional features AQ and SQ are practically equivalent but, curiously, it seems like the 128-dimensional features are more amenable to compression: for all compression rates the 128-dimensional features outperform the 1024-dimensional features ($[0.2646, 0.2293, 0.2101]$ vs. $[0.2917, 0.2562, 0.2246]$ in top-5 error), even though when uncompressed the 1024-dimensional features perform slightly better (0.1999 vs. 0.1893). This suggests that, if quantization is planned as part of a large-scale classification pipeline, low-dimensional features should be preferred over high-dimensional ones. It is also noticeable that for extreme compression rates (*e.g.*, 32 bits) PQ and OPQ have error rates in the 35-45% range, while AQ and SQ degrade

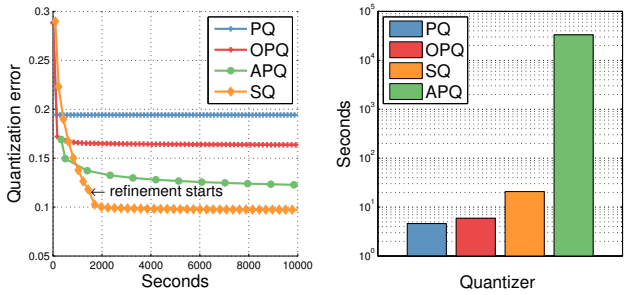

Figure 6. Left: Training time vs. Quantization error of the benchmarked methods on the ConvNet1M-128 training dataset (100K features). For clarity, we plot each 50 iterations for PQ and OPQ and each 25 iterations for SQ after initialization. PQ and OPQ complete 100 iterations after 286 and 336 seconds respectively (4.8 and 5.6 minutes), SQ takes $\sim$1500 seconds for initialization and $\sim$1000 seconds for 100 iterations of codebook refinement (42 minutes in total). APQ takes $\sim$2.7 hours for 10 iterations. Right: Encoding of the database set of 1M features. PQ and OPQ take $\sim$5 seconds, SQ $\sim$20 seconds, and APQ $\sim$9.2 hours.

more gracefully and maintain a 25-30% error rate.

4.4. Running times

Figure 6 shows the running time for training and database encoding for PQ/OPQ, APQ and SQ on the ConvNet1M-128 dataset using 8 codebooks (64 bits). All measurements were taken on a machine with a 3.20 GHz processor using a single core. We can see that SQ obtains most of its performance advantage out of initialization, but codebook refinement is still responsible for a 20% decrease to the final quantization error (0.12 to 0.10). We also see that APQ largely improves upon its OPQ initialization, but these iterations are extremely expensive compared to PQ/OPQ, and 3 iterations take almost as much computation as the entire SQ optimization. Beyond training (which arguably is not too big of a problem, since it only has to be done once), encoding the database with the learned codebooks is extremely expensive with APQ (9.2 hours), while for PQ/OPQ and SQ it stays in the 5-20 second range. Projecting these numbers to the encoding of a dataset with 1 billion features such as SIFT1B [11] suggests that PQ/OPQ would need about 1.5 hours to complete, and SQ would need around 6 hours; however, APQ would need around 1.05 years (!). Although all these methods are highly parallelizable, these numbers highlight the importance of fast encoding for good scalability.

5. Conclusions and future work

We have introduced Stacked Quantizers as an effective and efficient approach to compositional vector compression. After analyzing PQ and AQ in terms of their codebook assumptions, we derived a method that combines the best of both worlds, being only slightly more complex than PQ, while maintaining the representational power of AQ. We have demonstrated state-of-the-art performance on datasets of SIFT, GIST and, perhaps most importantly, deep convolutional features.

Future work will look at the integration of our pipeline with non-exhaustive indexing techniques such as the inverted file [11] or the inverted multi-index [2]. We also plan to investigate the use of optimization approaches that have proven useful in network-like architectures such as stochastic gradient descent and conjugate gradient.

Acknowledgements. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC, and enabled in part by WestGrid and Compute / Calcul Canada.

References

[1] R. Arandjelović and A. Zisserman. Multiple queries for large scale specific object retrieval. In *BMVC*, 2012.

[2] A. Babenko and V. Lempitsky. The inverted multi-index. In *CVPR*, 2012.

[3] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *CVPR*, 2014.

[4] A. Bergamo, L. Torresani, and A. W. Fitzgibbon. Picodes: Learning a compact code for novel-category recognition. In *NIPS*, 2011.

[5] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[6] K. Chatfield, K. Simonyan, and A. Zisserman. Efficient on-the-fly category retrieval using convnets and GPUs. *arXiv preprint arXiv:1407.4764*, 2014.

[7] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *AI*, 42(2), 1990.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[9] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *TPAMI*, 36(4), 2014.

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

[11] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33(1), 2011.

[12] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010.

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004.

[14] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, 2009.

[15] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, 2013.

[16] A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 155:23–36, 2006.

[17] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *ECCV*, 2010.

[18] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR*, 2011.

[19] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.

[20] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. *arXiv preprint arXiv:1311.2901*, 2013.