Additive Quantization for Extreme Vector Compression

Artem Babenko
Yandex, Moscow
Moscow Institute of Physics and Technology
arbabenko@yandex-team.ru

Victor Lempitsky
Skolkovo Institute of Science and Technology (Skoltech)
lempitsky@skoltech.ru

Abstract

We introduce a new compression scheme for high-dimensional vectors that approximates the vectors using sums of M codewords coming from M different codebooks. We show that the proposed scheme permits efficient distance and scalar product computations between compressed and uncompressed vectors. We further suggest vector encoding and codebook learning algorithms that can minimize the coding error within the proposed scheme. In the experiments, we demonstrate that the proposed compression can be used instead of or together with product quantization. Compared to product quantization and its optimized versions, the proposed compression approach leads to lower coding approximation errors, higher accuracy of approximate nearest neighbor search in the datasets of visual descriptors, and lower image classification error, whenever the classifiers are learned on or applied to compressed vectors.

1. Introduction

At least since the work [21], there is a growing interest in the computer vision community to the problem of extreme lossy compression of high-dimensional vectors. In this scenario, a large dataset of high-dimensional vectors (corresponding to image or interest point descriptors) is compressed by a large factor, so that only few bytes are spent to represent each vector. Computer vision applications additionally require that the compressed representation permits efficient evaluation of distances and/or scalar products between a query vector (which can be uncompressed) and the dataset of compressed vectors.

Previously proposed methods fall into two groups. The first very diverse group consists of binary encoding methods (e.g. [23, 9]) that transform each vector into a short sequence of bits, so that the Hamming distance between a pair of compressed vectors approximates the Euclidean distance between the original vectors. The second group is based on the idea of Product Quantization (PQ) [10]. PQ methods decompose each vector into components corresponding to orthogonal subspaces, and then vector quantize these lower-dimensional components using a separate small codebook for each subset. For some kinds of data (in particular, histogram-based descriptors such as SIFT) the orthogonal subspaces corresponding to natural dimension splitting lead to near optimal performance, and thus each of the PQ subsets corresponds to subsets of dimensions in the original space (Figure 1). For other types, a significant gain in PQ performance can be obtained by transforming the vectors via rotation found by an optimization process [15, 8].

Via a smart use of lookup tables, PQ compression permits very efficient computation of distances and scalar products between an uncompressed query vector and a large set of PQ-compressed vectors (*asymmetric distance computation* or ADC in terms of [10]). On top of the high efficiency of the ADC, the computed distances approximate the distances between uncompressed vectors rather closely, in particular, much more accurately than the Hamming distance approximation within binary encoding methods with the same compression rate – see e.g. comparisons in [15, 2]. PQ thus provides a unique combination of low approximation error, fixed size and random access to data on one hand, and highly efficient evaluation of distances and scalar products with uncompressed vectors on the other hand. At the same time, the decomposition into subspaces employed by PQ makes an underlying assumption that the distributions of vectors within different subspaces are mutually independent. The performance of PQ thus decreases whenever the dependence between subspace data distribution is strong.

In this paper we propose a new coding method called *additive quantization* (AQ) that generalizes PQ and further improves over PQ accuracy, while retaining its computational efficiency to a large degree. Similarly to PQ, AQ represents each vector as a sum of several components each coming from a separate codebook. Unlike PQ, AQ does not decompose data space into orthogonal subspaces and thus does not make any subspace independence assumptions. Thus the codewords within each AQ dataset are of the same length as the input vectors, and are generally not or-

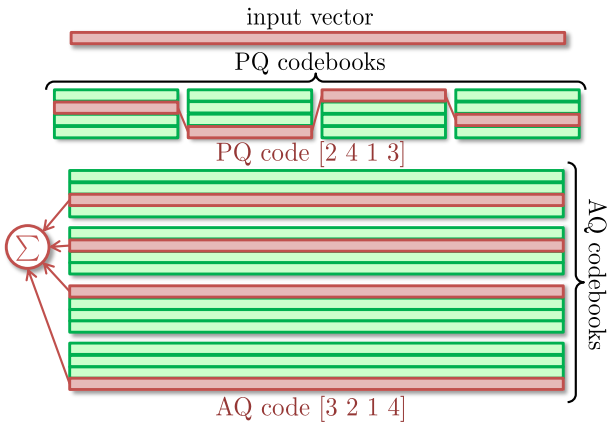


Figure 1. Product quantization (PQ) vs. Additive quantization (AQ) in the case of M=4 codebooks of size K=4. Both coding methods encode the input vector with M numbers between 1 and K. In the case of PQ, this code corresponds to the concatenation of M codewords of length D/M. In the case of AQ, this code corresponds to the sum of M codewords of length D. Given suitable codebooks, AQ is able to achieve better approximation to the input vector.

thogonal to each other. Consequently, unlike PQ, the codebooks in AQ are learned within a joint optimization process. As we demonstrate in the experiments with strongly compressed visual descriptors, removing independence assumptions allows AQ to attain better coding accuracy than PQ (Figure 1), which translates into higher accuracy of nearest neighbor search as well as of image classification, whenever the classifiers are learned on or are applied to compressed vectors.

Crucially, we show that similarly to PQ, AQ permits lookup table-based asymmetric distance and scalar product computations with uncompressed vectors. For small codebooks typically employed by PQ, the efficiency of scalar product computation with AQ is almost the same as in PQ. The distance computations (ADC) with AQ requires either small amount of extra time or small amount of extra memory compared to PQ, which in many applications would be justifiable by an improved accuracy. The encoding, the codebook learning, the distance computation, and the improvement over PQ are all particularly favourable for short codes (e.g. four or eight bytes), i.e. extreme compression rates. For longer codes, the two techniques (AQ and PQ) can be seamlessly combined.

Apart from PQ, AQ is connected to several other frameworks popular in computer vision and general pattern analysis. In particular, AQ can be regarded as a very special case of sparse coding with unit component weights. Below, we also show interesting connections between AQ and Markov Random Field optimization. Finally, AQ can be regarded as a generalization of standard vector quantization, and the codebook learning algorithm that we propose for AQ is a generalization of the k-means algorithm.

In the remainder of the paper, we discuss the representation used by AQ, the lookup table-based computation of scalar products and distances, the encoding algorithms for AQ representation, and the codebook learning. We then perform extensive experimental comparison between AQ, PQ (including the optimized version [15]) in the context of nearest-neighbor search and image classification.

2. Additive quantization

2.1. Additive quantization representation

We now introduce notation and discuss additive quantization (AQ) in detail. Below, we assume that we deal with D-dimensional vectors. AQ is based around a set of M codebooks, each containing K vectors (codewords). We denote the mth codebook as C^m, and the kth codeword in the mth codebook as $c^m(k)$. This setting is similar to PQ. However, whereas in PQ the codewords have the length D/M, the length of codewords in AQ are D, i.e. they have the same dimensions as the vectors that are being encoded.

The AQ model encodes a vector $x \in \mathbf{R}^D$ as a sum of M codewords (one codeword per codebook). In more detail, a vector is encoded with an M-tuple of codeword IDs $[i_1, i_2, \ldots, i_M]$, where each ID is between 1 and K. The encoding process (described below) seeks the code that minimizes the distance between x and the sum of the corresponding codewords:

$$x \approx \sum_{m=1}^{M} c^m(i_m), \quad i_m \in 1..K \tag{1}$$

If, for example $K = 256$ (as in most of our experiments), then the vector is encoded into M bytes, each byte encoding a single codeword ID. The memory footprint of an AQ-encoded vector will be the same as a PQ-encoded vector (for the same M and K), while AQ can potentially represent the vector more accurately. The codebooks within AQ occupy M times more memory than within PQ, however this memory increase is typically negligible for datasets that are large enough.

2.2. Fast distance and scalar product computations

The PQ compression permits very efficient computation between an uncompressed vector (e.g. a query q to a nearest-neighbor search) and a large number $L \gg K$ of PQ-compressed vectors, so that each distance evaluation is implemented with M lookups and $M-1$ additions, plus a small amount of precomputation independent of L. Such asymmetric distance computation (ADC) is arguably where the main power of PQ lies. We will now show that with some caveats all this is possible in the case of AQ.

We now assume that we need to compute squared euclidean distances between the query q and the dataset of L AQ-encoded vectors. We start with the formula:

$$\|q - x\|^2 = \|q\|^2 - 2\langle q, x\rangle + \|x\|^2 \tag{2}$$

We can precompute and reuse $\|q\|^2$ for each of L dataset vectors, so the two questions that remain are how to evaluate $\langle q, x\rangle$ and $\|x\|^2$ quickly. We now assume that x in (2) is AQ-compressed, i.e. $x = \sum_{m=1}^{M} c^m(i_m)$.

Evaluating the scalar product. The evaluation of $\langle q, x\rangle$ can be implemented rather straightforwardly using lookup tables. Indeed,

$$\langle q, x\rangle = \sum_{m=1}^{M} \langle q, c^m(i_m)\rangle = \sum_{m=1}^{M} T^m(i_m)\ , \tag{3}$$

where $T^m(i_m) = \langle q, c^m(i_m)\rangle$ can be precomputed and stored, given the query q. Given the query and L AQ-encoded vectors, the total complexity involved will be $O(DMK)$ to compute the lookup tables (versus $O(DK)$ in an analogous step of the PQ) and $O(ML)$ to compute the actual scalar products. Assuming that $L \gg DK$ as is typical in many applications related to the nearest neighbor search, the complexity of the scalar product computation within AQ is very similar to the complexity of distance computation in the case of PQ.

Evaluating the $\|x\|^2$. Evaluating $\|x\|^2$ can also be facilitated by the lookup tables. Indeed:

$$\|x\|^2 = \|\sum_{m=1}^{M} c^m(i_m)\|^2 = \sum_{m=1}^{M}\sum_{m'=1}^{M} \langle c^m(i_m), c^{m'}(i_{m'})\rangle\ , \tag{4}$$

Each of the individual terms in the right-hand side can be precomputed and stored in the lookup table (note that the terms are query independent). The number of operations required to compute $\|x\|^2$ is therefore approximately $M^2/2$ lookups and $M^2/2$ additions (assuming that the symmetry of the scalar product is exploited).

Notably, the cost of computation of $\|x\|^2$ is independent of the space dimensionality D. However, it grows quadratically with M and can slow down the computation of the distances considerably. It is possible to get rid of this time overhead at the cost of the small memory overhead using the fact that the term $\|x\|^2$ does not depend on the query q. For this, we can encode the squared length $\|x\|^2$ using a single byte by non-uniformly quantizing such scalar values over the encoded (or a hold-out) dataset. At the data compression time, we then augment the AQ-code of each x with the corresponding *length byte*. The computation of $\|x\|^2$ in this case costs one lookup (and one byte per vector). In our experiments we found that the effect of length quantization on the accuracy of the nearest neighbor search is minimal, which is natural to expect given that we quantize a scalar value.

Finally, we note that for some applications, e.g. when applying a linear classifier to a large set of compressed vectors, the only required operation is scalar product computation (between the weight vector and the encoded vector), and therefore the computation of $\|x\|^2$ is unnecessary.

2.3. Vector encoding

We now come to the task of vector encoding, i.e. finding the AQ-representation for a vector x given the codebooks $C^1 \dots C^M$. We seek the code that minimizes the coding error E:

$$E(i_1, i_2, \dots i_m) = \|x - \sum_{m=1}^{M} c^m(i_m)\|^2 \tag{5}$$

Using (2), the error function can be rewritten as:

$$E(i_1, i_2, \dots i_m) = \sum_{m=1}^{M} \left[-2\langle x, c^m(i_m)\rangle + \|c^m(i_m)\|^2\right] + \sum_{1 \le m < m' \le M} \left[2\langle c^m(i_m), c^{m'}(i_{m'})\rangle\right] + \|x\|^2 \tag{6}$$

Given x, the term $\|x\|^2$ is constant, while the terms $U_m(i_m) = -2\langle x, c^m(i_m)\rangle + \|c^m(i_m)\|^2$ can be precomputed and stored. The terms $V_{m,m'}(i_m, i_{m'}) = 2\langle c^m(i_m), c^{m'}(i_{m'})\rangle$ can be precomputed independently of the query. After such precomputations (and after omitting the constant), the error function (6) can be rewritten as:

$$E(i_1, i_2, \dots i_m) = \sum_{m=1}^{M} U_m(i_m) + \sum_{1 \le m < m' \le M} V_{m,m'}(i_m, i_{m'})\ , \tag{7}$$

i.e. as a fully connected discrete pairwise MRF energy. Importantly for any optimization algorithm, it can be evaluated at the computational cost independent of the space dimensionality D. The optimization problem can be solved approximately by any of the existing algorithms like Loopy Belief Propagation (LBP) [17], Iterative Conditional Models (ICM)[6], etc. However, because of full connectivity and the general form of the pairwise potentials, we observed that LBP and ICM, and, in fact other algorithms from the MRF optimization library [12] perform poorly.

Instead, we propose another approximate algorithm which constructs the output tuple successively. We adapt the idea of a general Beam Search algorithm [20] to our particular problem. The resulting algorithm is also reminiscent of matching pursuit used in sparse coding [14]. In our case

Beam Search starts its work by finding N elements which are the closest to the vector x from the $C{=}C_1 \cup \ldots \cup C_M$. These elements become seeds for tuples which are candidates to result in the best approximation. Those N incomplete tuples are maintained through next $M{-}1$ iterations. At the iteration m ($m{>}1$), Beam Search considers each incomplete tuple, containing $m{-}1$ vectors from $m{-}1$ codebooks. It then considers the remaining $M{+}1{-}m$ codebooks and finds the N codewords among all codewords in those codebooks that are closest to the remainder of x after subtracting codewords already included into the tuple. Thus, after considering all N candidates, N^2 tuples of length m are created. We then pick the top N unique tuples (in terms of the approximation error for x) and keep them as candidates for the next iteration. After M such iterations, i.e. when tuples contain exactly M elements, a tuple with the best approximation error is returned.

The lookup tables are used through Beam Search to make all operations (except for the precomputations of these tables) independent of the space dimensionality D. We observe that for reasonable N (e.g. $N{=}16$ or $N{=}32$), Beam Search performs much better then other MRF-optimization algorithms we tried (given comparable amount of time). Therefore, we use Beam Search in our experiments. Interestingly, we tried to formulate the MRF optimization (7) as an integer quadratic program and submit it to a general-purpose branch-and-cut optimizer [1]. For small codebooks $K{=}64$ and $M = 8$ and given very large amount of time, the solver could find a global minimum with much smaller energy (coding error) then those found by Beam Search or other non-exhasutive algorithms. While such "smart brute-force" approach is infeasible for large datasets and meaningful codebook sizes, this result suggests the AQ-coding problems as interesting instances for the research into new MRF-optimization algorithms.

2.4. Codebook learning

We finally consider the task of learning codebooks, i.e. finding a set of M codebooks $\{C^1, C^2, \ldots C^M\}$ that can encode a vector set $X = \{x_1, \ldots, x_n\}$ with low coding error. Thus, we seek to minimize:

$$\min_{\substack{C^1,\ldots,C^M \subset \mathbf{R}^D \\ |C^m|=K \\ i_j^m \in 1..K}} \sum_{j=1}^{n} \|x_j - \sum_{m=1}^{M} c^m(i_j^m)\|^2 \qquad (8)$$

Similarly to other codebook learning approaches, we perform minimization using block-coordinate descent, alternating the minimization over the assignment variables (codes) i_j^m and the codewords $c^m(\cdot)$. The proposed algorithm generalizes the standard k-means algorithm (which corresponds to the case $M{=}1$).

The minimization over the encoding variables given codebooks requires coding the vectors x_j given the codebooks, which is covered in the previous subsection. Updating the codewords given assignments is equivalent to the following least-squares problem:

$$\min_{\{c^m(k)\}} \sum_{j=1}^{n} \|x_j - \sum_{m=1}^{M} \sum_{k=1}^{K} a_{km}^j c^m(k)\|^2 \qquad (9)$$

$$a_{km}^j = \begin{cases} 1, & \text{if } i_j^m{=}k \\ 0, & \text{otherwise} \end{cases}$$

While the least-squares optimization (9) may seem large (given large n and D), one can observe that it decomposes over each of the D dimensions. Thus, instead of solving a single large-scale least squares problem with KMD variables, it is sufficient to solve D least-squares problems with KM variables, which can be formulated as an overconstrained system of linear equations:

$$\forall j = 1..n \quad \sum_{m=1}^{M} \sum_{k=1}^{K} a_{km}^j c^m(k)_d = x_{j,d} \ , \qquad (10)$$

where $c^m(k)_d$ is the dth component of $c^m(k)$, $x_{j,d}$ is the dth component of x_j. (10) defines n equations over KM variables, which are solved in the least-quadratic sense. As an additional optimization, one can note that the D overconstrained systems (10) for different dimensions differ only in the right-hand side, so that the left-hand side sparse matrix can be stored and reused for all dimensions. As a result, the complexity of codebook learning is dominated by the encoding step.

In most of the experiments below, we initialize learning process by setting the assignment variables to random numbers between 1 and K. It is also possible to perform initialization by codebooks obtained within PQ or OPQ (a PQ codeword can be turned into a full-length AQ vector, by padding it with zero chunks). Experimentally, we have observed that the final results were insensitive to the variations in the codebook initialization.

2.5. Additive product quantization

The complexity of the Beam Search algorithm grows cubically with M. While for the extreme compression (e.g. $M{=}4$) the encoding complexity can be well acceptable even for large datasets, for larger M more efficient encoding algorithms are required, and we leave this for the future work. In the meantime, as long as the less extreme compression into a large number of bytes is desired, this can be achieved using a hybrid approach that combines PQ and AQ. In this approach, the vector can be split into M_1 orthogonal components (as in PQ), whereas each component can then be encoded by AQ into M_2 bytes. One can therefore keep M_2 small and AQ encoding efficient, while using $M_1{\times}M_2$

bytes for the coding of each vector. As we show in the experiments, such hybrid compression is more accurate than the PQ compression with $M = M_1 \times M_2$.

3. Experiments

In this section, we provide the results for a number of comparative evaluations of the proposed approach (additive quantization) as well as the product quantization [10], and one of the versions of the optimized product quantization (OPQ) called *Cartesian K-Means* [15] (using the authors implementation). These PQ-based methods represent the state-of-the-art in vector compression, and have been shown to perform better than binary encoding methods in a number of recent comparisons [15, 3].

We focus our attention on small codebook sizes ($K = 256$) and extreme compression levels (M=8 and M=16). For the codes of length M=8 and longer, we also consider the hybrid algorithm (Additive Product Quantization) that uses OPQ optimization to rotate the data and then applies AQ encoding (into 4 bytes) to different parts of the rotated vector (e.g. the halves in the case of 8 byte codelength, or quarters in the case of 16 byte codelength). Throughout the experiments, all codebooks and rotation matrices are learned on the hold-out sets. In all experiments, we set the parameter N within the Beam Search to 16 during codebook learning and to 64 during the data encoding.

We start by comparing the approximation error between the methods. In Figure 2 we plot the mean approximation error as a function of the code length for the well-known SIFT1M dataset [10] containing one million 128-dimensional SIFT vectors (plus the holdout set). For all code lengths (M=4, 8, 16) AQ/APQ approximation error is considerably lower than for the current state-of-the-art methods PQ and OPQ. Given these encouraging results, we consider two applications where more accurate coding may result in better performance, namely (i) approximate nearest neighbor search and (ii) image classification with limited memory.

We also perform the comparison of AQ and OPQ approximation quality for the varying codebook sizes K. Figure 3 shows that AQ provides substantially better compression for the entire range of K.

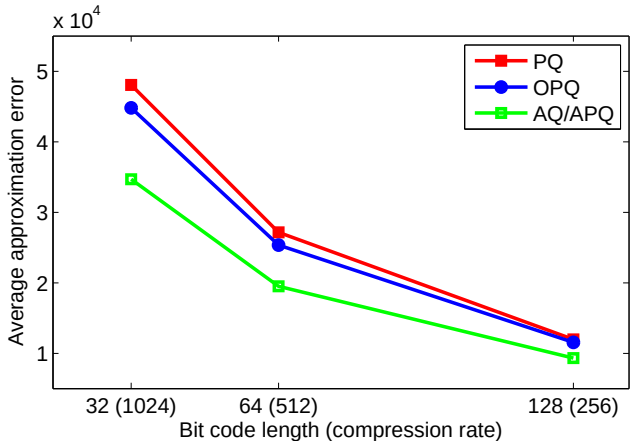


Figure 2. Approximation errors on SIFT-1M dataset for different compression methods and three code lengths (4, 8, 16 bytes). Additive quantization (AQ) was used for 4 bytes, Additive product quantization (APQ) was used for 8 and 16 bytes. For all code lengths, the error of AQ/APQ is lower than for Product Quantization (PQ) and Optimized Product Quantization (OPQ).

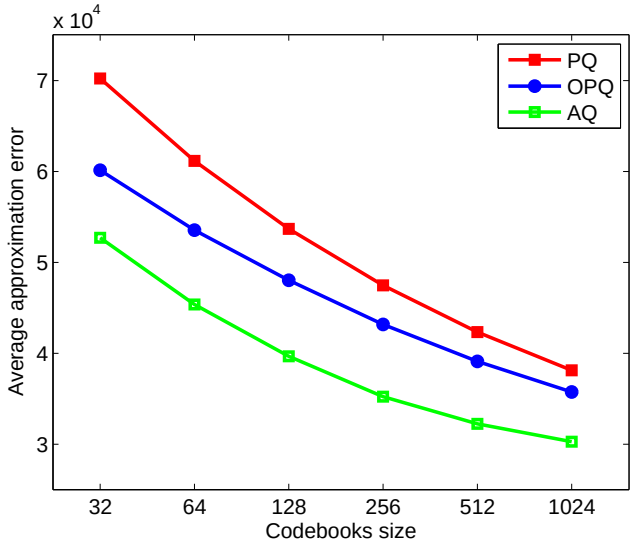


Figure 3. Approximation errors on 10,000 SIFT-descriptors for AQ and OPQ with different codebooks sizes (4 codebooks are used in each case). The advantage of AQ is uniform.

3.1. Nearest neighbor (NN) search

Approximate NN-search is a technology used widely in pattern analysis and computer vision in particular. It is also a natural testbed to compare different encoding methods. Here, we perform a comparison on three datasets of visual descriptors described below. The datasets are compressed using the compared methods. We then consider several queries (given by uncompressed vectors) for which the true Euclidean nearest neighbor in the dataset ("ground truth") is precomputed. For each query evaluate the distance between the query and each of the compressed vectors in a dataset. After reranking, we report the *recall@T* measure [10], defined as a probability (computed over a number of queries) that the set of T closest compressed vectors contains the true nearest neighbor. The results are shown in the form of recall@T-vs-T curves.

The considered datasets are briefly discussed below.

SIFT-1M: This dataset introduced in [10] contains one million of 128-dimensional SIFT descriptors [13], plus 100,000 descriptors in a hold out set. It also comes with

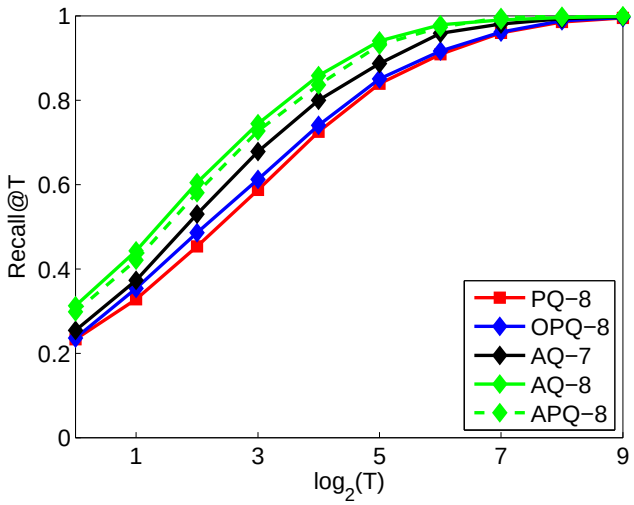


Figure 5. Comparison of different encodings with the code length $M = 8$ bytes. AQ-8 has the highest recall, APQ-8 and AQ-7 are slightly less accurate but are much faster then AQ-8 in terms of distance evaluation, see Table 1. The AQ-7 method is also faster than PQ and OPQ, while achieving higher retrieval accuracy.

10,000 queries with known true Euclidean nearest neighbors (among the main dataset).

GIST-1M: This dataset introduced in [10] contains one million of 960-dimensional GIST descriptors [16] in the main set, and 500,000 vectors in the hold-out set. The ground truth (true Euclidean nearest neighbors) is known for a hold-out set of 1,000 queries.

VLAD-500K: This dataset contains VLAD descriptors [11] of natural images. The descriptors are PCA-compressed (with whitening) to 128 dimensions. The base and the learn sets both contain 500,000 vectors, the query set contains 1,000 vectors for which we precomputed the ground truth neghbors in the base set[1].

The SIFT and GIST datasets thus contain descriptors based on the spatially-binned histograms of gradients, while the PCA-compressed VLAD descriptors have a more complex structure. On the other hand, SIFT and VLAD datasets have the same dimensionality $D = 128$, while GIST descriptors are more high-dimensional.

The Figure 4 shows relative performance of all the methods (AQ, PQ, OPQ) for the case of extreme vector compression to four bytes. AQ demonstrates an advantage over the other methods for all datasets and all T, with the advantage being the largest for the SIFT descriptors.

We further investigate the performance on the SIFT dataset, now considering the case of $M = 8$ bytes. Here, along the standard AQ we consider two more variants: APQ (the hybrid algorithm) and *AQ-7*, where we use $M = 7$ bytes to encode each vector and we spend one more byte

[1]We thank Dr. Relja Aranjelović for providing the dataset.

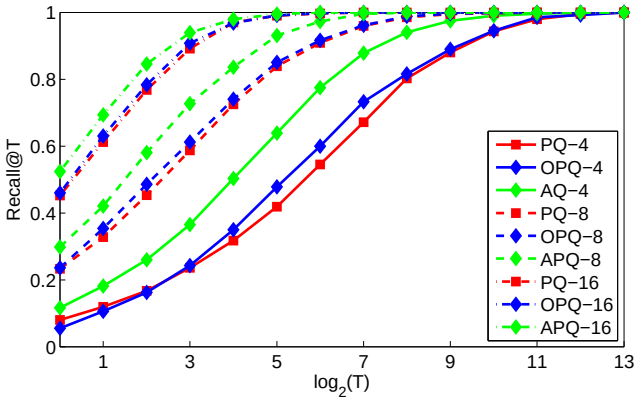


Figure 6. Advantage of AQ/APQ over PQ and OPQ for different code lenghts. The advantage of AQ methods is particularly large in the case of shorter code lengths.

to encode the length of the compressed vector (as discussed in the end of Section 2.1). The latter has the same memory footprint as PQ/OPQ **and** a lower cost of the distance evaluation (Table 1). As can be seen (Figure 5) the variants of the AQ (including the AQ-7 variant) again outperform the PQ and OPQ methods.

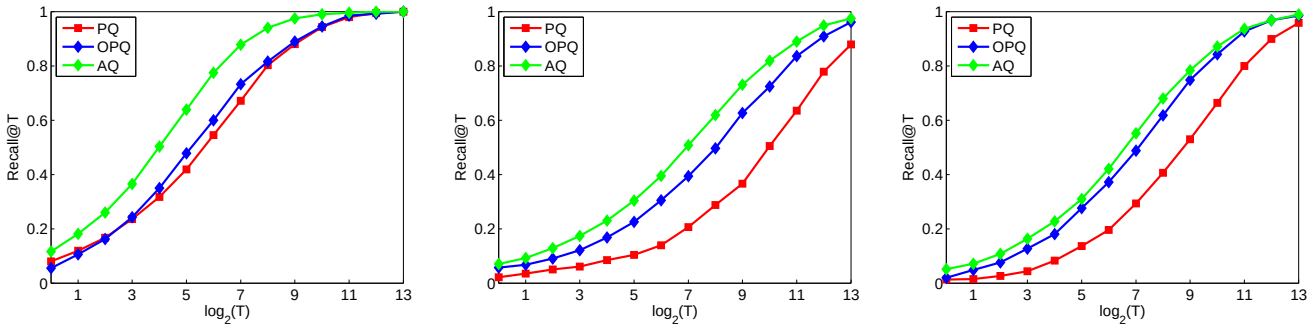
Finally, Figure 6 shows alongside the curves corresponding to PQ, OPQ and AQ/APQ correspong to different code length (AQ is used for $M = 4$ bytes and APQ for $M = 8$ and $M = 16$ bytes). As expected, all methods improve their accuracies for longer codes, with the methods based on additive quantization performing better than OPQ and PQ. The difference in the performance between the methods decreases when M increases.

3.2. Classification

Another application of compact encoding is a compression of image descriptors for image classification. There are two scenarios where it can be used. Firstly, image compression can be applied to training images in order to facilitate learning on very large image collections when the whole training data needs to be loaded into the CPU [19, 22].

The second, and perhaps more important scenario, is the compression of the test set. Here a classifier can be trained on a smaller set of images and then applied to a very large dataset of compressed image descriptors in order to find the images with the highest classification score [5, 4, 2]. Interestingly, in this scenario it is only necessary to evaluate the scalar products between the query (the classifier) and the compressed vectors. Since the evaluation of the $\|x\|^2$-term is not necessary, the search is equally efficient in the case of AQ and PQ (save for the lookup table precomputation) even without the use of an extra byte encoding the vector length.

We have conducted the experiments corresponding to both scenarios on the PASCAL VOC 2007 [7]. We used



SIFT-1M dataset **GIST-1M dataset** **VLAD-500K dataset**

Figure 4. Euclidean NN recall@T (number of items retrieved) based on different compression methods for SIFT-1M, GIST-1M, and VLAD-500K datasets with the code length M=4 bytes. For all datasets the recall@T with the AQ encoding is higher than for with the PQ and the OPQ encodings for all values of T.

Fisher Vector descriptors[18] with 256 components over SIFT descriptors PCA-ed to 80 components. We do not use the spatial pyramids. Our goal was to evaluate the degradation from the use of OPQ and APQ for different compression rates. In both compression methods, original Fisher Vectors are split into R subvectors and each subvector is compressed either by OPQ or by AQ into $M = 4$ bytes (which gives the compression rate R). Different compression rates can be obtained via varying R. We use the standard mean average precision measure for PASCAL classification experiments.

Experiment 1: compression of training data. In this experiment, we used the PASCAL test set to learn the codebooks/rotation matrices. We then learned the classifiers on the train+val part of the data using either uncompressed vectors or APQ and OPQ compressed vectors (which were decompressed during learning as in [19]). Once we learned the classifiers, we applied them to the uncompressed test set and measure the standard performance (average precision). As can be seen from Figure 7, the degradation is smaller in the case when the training data is APQ compressed.

Experiment 2: compression of test data. In this experiment, we followed the second scenario and learned classifiers using the uncompressed descriptors on the PASCAL train+val set. We evaluated the effect of the compression of the test data (the compression scheme was the same as in the Experiment 1, AQ/OPQ codebooks were learned on the PASCAL train data). Once again, from Figure 8 we could see that the degradation resulting from the compression is smaller in the case of the APQ compression.

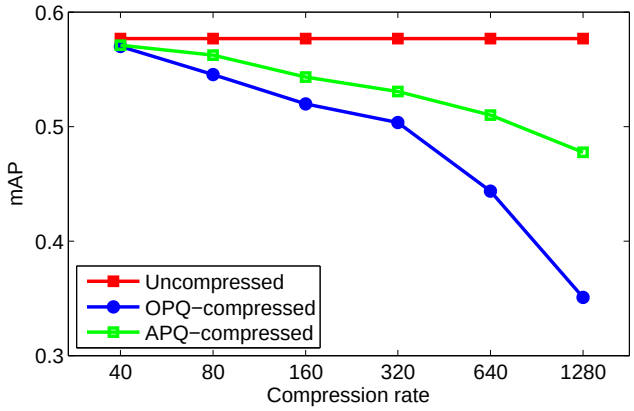


Figure 7. Average precision of classification for **learning on compressed data** and testing on uncompressed data. Codebooks for OPQ and APQ were learned on test set and were used to encode the training and validation sets. The classifiers were learned on the training and validation sets and were tested on the test set. Better coding approximation of the APQ results in higher classification accuracy.

Method	OPQ	AQ	APQ	AQ7
Increase w.r.t PQ, 4 bytes	1.05	2.55	—	—
Increase w.r.t PQ, 8 bytes	1.05	5.46	2.59	0.92

Table 1. The NN-search runtime increase relative to PQ for different encoding methods (code length M=4 and M=8 bytes). The PQ runtime is assumed to equal one. In the case of M=8 bytes, the AQ-7 method is the fastest (and also outperforms PQ and OPQ in terms of accuracy). AQ and APQ are more accurate but slower then AQ-7. Please see text for more details

4. Discussion

We have introduced the new lossy compression scheme for high-dimensional vectors that permits efficient computation of scalar products and distances based on lookup tables. The scheme provides higher approximation accuracy than the previous state-of-the-art (optimized product quantization), which translates into higher accuracy of approximate nearest neighbor search and higher classification accuracies in image classification applications, whenever extreme compression is applied to the training or to the test data.

In terms of speed, AQ is almost as fast as OPQ when

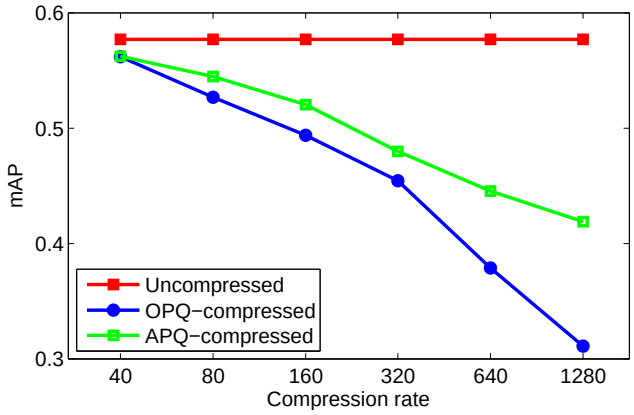


Figure 8. Average precision of classification for learning on uncompressed data and **testing on compressed data**. The classifiers were learned on the training and validation sets and were tested on the test set. Better coding approximation of the AQ results in higher classification accuracy.

evaluating the scalar products. Evaluating distances is slower due to the need to evaluate the $\|x\|^2$ term (Table 1). This can be alleviated if this length is encoded using a separate byte, in which case the evaluation of the distance becomes as fast as in the case of PQ. This extra byte can be allocated out of the total budget for memory footprint, e.g. in Figure 5 we used seven out of eight bytes to store the AQ-compressed vector and the remaining byte to store the length. In this case, the system based on AQ outperformed the system based on OPQ both in terms of accuracy and in terms of speed of distance evaluation, while using the same amount of memory per vector.

Currently, the main limitation of the proposed scheme is the complexity of the vector encoding step, and this the direction in which further performance gains could be possible. Such improvement should be particularly considerable for the case of longer codes, for which we currently use the hybrid algorithm combining PQ and AQ. The current hybrid scheme results in the fully connected graph in (7) being effectively replaced with disjoint cliques of size four. We are currently investigating other approximations to the fully-connected graphs (e.g. based on Chow-Liu trees).

References

[1] Gurobi optimizer. `http://www.gurobi.com/`, 2013. 4

[2] R. Arandjelović and A. Zisserman. Multiple queries for large scale specific object retrieval. In *British Machine Vision Conference*, 2012. 1, 6

[3] R. Arandjelović and A. Zisserman. Extremely low bit-rate nearest neighbor search using a Set Compression Tree. Technical report, Department of Engineering Science, University of Oxford, 2013. 5

[4] A. Bergamo and L. Torresani. Meta-class features for large-scale object categorization on a budget. In *CVPR*, pages 3085–3092, 2012. 6

[5] A. Bergamo, L. Torresani, and A. W. Fitzgibbon. Picodes: Learning a compact code for novel-category recognition. In *NIPS*, pages 2088–2096, 2011. 6

[6] J. Besag. *On the Statistical Analysis of Dirty Pictures*. J. Roy. Stat. Soc. B, 1986. 3

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 6

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

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

[10] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33(1), 2011. 1, 5, 6

[11] H. Jegou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010. 6

[12] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, J. Lellmann, N. Komodakis, and C. Rother. A comparative study of modern inference techniques for discrete energy minimization problems. In *CVPR*, pages 1328–1335, 2013. 3

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004. 5

[14] S. Mallat and Z. Zhang. Matching pursuit in a time-frequency dictionary. *IEEETransactions on Signal Processing*, 41(12):33973415, 1993. 3

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

[16] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3), 2001. 6

[17] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988. 3

[18] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *ECCV*, 2010. 7

[19] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR*, pages 1665–1672, 2011. 6, 7

[20] S. C. Shapiro. *Encyclopedia of Artificial Intelligence*. 1987. 3

[21] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008. 1

[22] A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *CVPR*, pages 2320–2327, 2012. 6

[23] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008. 1