

SEARCHING IN ONE BILLION VECTORS: RE-RANK WITH SOURCE CODING

Hervé Jégou
INRIA Rennes

Romain Tavenard
University Rennes I

Matthijs Douze
INRIA Grenoble

Laurent Amsaleg
CNRS, IRISA

ABSTRACT

Recent indexing techniques inspired by source coding have been shown successful to index billions of high-dimensional vectors in memory. In this paper, we propose an approach that re-ranks the neighbor hypotheses obtained by these compressed-domain indexing methods. In contrast to the usual post-verification scheme, which performs exact distance calculation on the short-list of hypotheses, the estimated distances are refined based on short quantization codes, to avoid reading the full vectors from disk.

We have released a new public dataset of one billion 128-dimensional vectors and proposed an experimental setup to evaluate high dimensional indexing algorithms on a realistic scale. Experiments show that our method accurately and efficiently re-ranks the neighbor hypotheses using little memory compared to the full vectors representation.

Index Terms— nearest neighbor search, quantization, source coding, high dimensional indexing, large databases

1. INTRODUCTION

Approximate nearest neighbors (ANN) search methods [3, 10, 14, 15] are required to handle large databases, especially for computer vision [12] and music retrieval [2] applications. One of the most popular techniques is Euclidean Locality-Sensitive Hashing [3]. However, most of these approaches are memory consuming, since several hash tables or trees are required. The methods of [4, 15], which embeds the vector into a binary space, better satisfies the memory constraint. They are, however, significantly outperformed in terms of the trade-off between memory usage and accuracy by recent methods that cast high dimensional indexing to a source coding problem [11, 5, 6], in particular the product quantization-based method of [5] exhibits impressive results for large scale image search [6].

State-of-the-art approaches usually perform a re-ranking stage to produce a ranked list of nearest neighbors. This is always done in partitioning based method such as LSH [3] or FLANN [10], as the neighbor hypotheses are not ranked on output of the index. But as shown in [5], this post verification is also important for methods based on binary [4, 15] or quantized codes [11, 5, 6], as the ranking provided on output of the large scale search is significantly improved when verifying the few first hypotheses. In all these approaches, re-ranking is performed by computing the exact Euclidean distance between each query and the returned hypotheses. For large datasets, this raises a memory issue: the vectors can not be stored in central memory and must be read from disk on-the-fly. Moreover, the vectors are to a large extent accessed randomly, which in practice limits the number of hypotheses that can be verified.

In this paper, we propose a new post-verification scheme in the context of compression-based indexing methods. We focus on the method presented in [5], which offers state-of-the-art performance, outperforming the FLANN which was previously shown to outperform LSH [10]. It also provides an explicit approximation of the

indexed vectors. Our algorithm exploits the approximation resulting from the first ranking, and refines it using codes stored in RAM. There is an analogy between this approach and the scalable compression techniques proposed in the last decade [13], where the term “scalable” means that a reconstruction of the compressed signal is refined in an incremental manner by successive description layers.

In order to evaluate our approach, we introduce a dataset of one billion vectors extracted from millions of images using the standard SIFT descriptor [8]. Testing on a large scale is important, as most ANN methods are usually evaluated on sets of unrealistic size, thereby ignoring memory issues that arise in real applications, where billions of vectors have to be handled [4, 9]. The groundtruth nearest-neighbors have been computed for 10000 queries using exact distance computations. To our knowledge, this set is the largest ever released to evaluate ANN algorithms against an exact linear scan on real data: the largest other experiment we are aware of is the one reported in [7], where a private set of 179 million vectors was considered. [1] reports an experiment on 1 billion vectors, but on synthetic data with a known model exploited by the algorithm.

Experiments performed on this dataset show that the proposed approach offers an alternative to the standard post-verification scheme. The precision of the search is significantly improved by the re-ranking step, leading to state-of-the-art performance on this scale, without accessing the disk.

2. CONTEXT: COMPRESSION BASED INDEXING

In this section, we briefly review the indexing method of [5], which finds the approximate k nearest neighbors using a source coding approach. For the sake of presentation, we describe only the Asymmetric Distance Computation (ADC) method proposed in [5]. We also assume that we search for the nearest neighbor (i.e., $k = 1$),

Let $x \in \mathbb{R}^d$ be a query vector and $\mathcal{Y} = \{y_1, \dots, y_n\}$ a set of vectors in which we want to find the nearest neighbor $\text{NN}(x)$ of x . The ADC approach consists in encoding each vector y_i by a quantized version $c_i = q_c(y_i) \in \mathbb{R}^d$. For a quantizer $q_c(\cdot)$ with K centroids, the vector is encoded by $b_c = \log_2(K)$ bits, assuming K is a power of 2. An approximate distance $d_c(x, y_i)$ between a query x and a database vector is computed as

$$d_c(x, y_i)^2 = \|x - q_c(y_i)\|^2. \quad (1)$$

The approximate nearest neighbor $\text{NN}_a(x)$ of x is obtained by minimizing this distance estimator:

$$\text{NN}_a(x) = \arg \min_i d_c(x, y_i)^2 = \arg \min_i \|x - q_c(y_i)\|^2, \quad (2)$$

which is an approximation of the exact distance calculation

$$\text{NN}(x) = \arg \min_i \|x - y_i\|^2. \quad (3)$$

Note that, in contrast with the binary embedding method of [15], the query x is not converted to a code: there is no approximation error on the query side.

To get a good vector approximation, K should be large ($K = 2^{64}$ for a 64 bit code). For such large values of K , learning a K -means codebook is not tractable, neither is the assignment of the vectors to their nearest centroids. To address this issue, [5] uses a product quantizer, for which there is no need to explicitly enumerate the centroids. A vector $y \in \mathbb{R}^d$ is first split into m subvectors $y^1, \dots, y^m \in \mathbb{R}^{d/m}$. A product quantizer is then defined as a function

$$q_c(y) = (q^1(y^1), \dots, q^m(y^m)), \quad (4)$$

which maps the input vector y to a tuple of indices by separately quantizing the subvectors. Each individual quantizer $q^j(\cdot)$ has K_s reproduction values, learned by K -means. To limit the assignment complexity, $\mathcal{O}(m \times K_s)$, K_s is set to a small value (e.g. $K_s=256$). However, the set of K centroids induced by the product quantizer $q_c(\cdot)$ is large, as $K = (K_s)^m$. The squared distances in Equation 2 are computed using the decomposition

$$d_c(x, y)^2 = \|x - q_c(y)\|^2 = \sum_{j=1, \dots, m} \|x^j - q^j(y^j)\|^2, \quad (5)$$

where y^j is the j^{th} subvector of y . The squared distances in the sum are read from look-up tables. These tables are constructed on-the-fly for a given query, prior to the search in the set of quantization codes, from each subvector x^j and the k_s centroids associated with the corresponding quantizer q^j . The complexity of the table generation is $\mathcal{O}(d \times K_s)$. When $K_s \ll n$, this complexity is negligible compared to the summation cost of $\mathcal{O}(d \times n)$ in Equation 2.

This approximate nearest neighbor method implicitly sees multi-dimensional indexing as a vector approximation problem: a database vector y can be decomposed as

$$y = q_c(y) + r(y), \quad (6)$$

where $q_c(y)$ is the centroid associated with y and $r(y)$ the error vector resulting from the quantization, called the *residual* vector. It is proved [5] that the square error between the distance and its estimation is bounded, on average, by the quantization error. This ensures, asymptotically, perfect search results when increasing the number of bits allocated for the quantization indexes.

The ADC indexing method is fully parametrized by the number of subvectors m and the total number of bits b_c . In the following, we set $b_c = 8$ (i.e., $K_s = 256$), as suggested in [5], which means that we use exactly m bytes per indexed vector.

3. RE-RANKING NEIGHBORS USING SOURCE CODING

3.1. Refinement: principle

The objective of the method proposed in this paper is to avoid the costly post-verification scheme adopted in most state-of-the-art approximate search techniques [3, 10]. The idea is to take advantage of the information on the database point provided by the indexing. This is possible when using the ADC method [5]: this search algorithm provides an explicit approximation $q_c(y)$ of database vector y .

We first assume that the first retrieval stage returns a set of k' hypotheses. These vectors are the one for which a post-verification is required. For each database vectors y_i , the error vector is equal to

$$r(y) = y - q_c(y). \quad (7)$$

The proposed method consists in reducing the energy of this residual vector to limit the impact of the approximation error on the

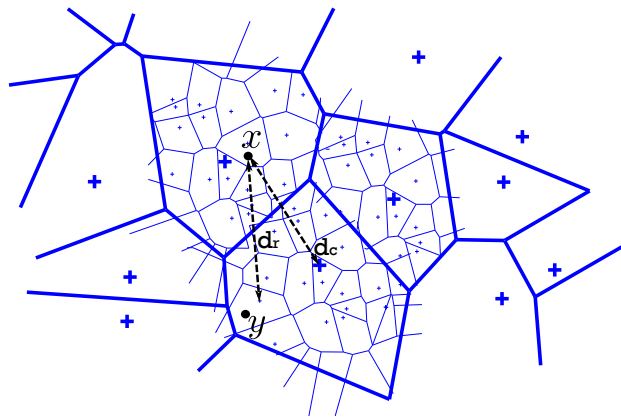


Fig. 1. Illustration of the proposed refinement process. For each database vector y , the distance $d_c(x, y) = d(x, q_c(y))$ is computed to build the short-list of potential nearest neighbors. For selected y vectors, the distance is re-estimated by $d_r(x, y)$, which is obtained by computing the distance between y and its improved approximation $d_r = q_c(y) + q_r(y - q_c(y))$.

estimated distances. This is done by encoding the residual vector $r(y)$ using another product quantizer q_r defined by its reproduction values \mathcal{C}_r :

$$q_r(r(y)) = \arg \min_{c \in \mathcal{C}_r} \|r(y) - c\|^2, \quad (8)$$

where the product quantizer $q_r(\cdot)$ is learned on an independent set of residual vectors. Similar to $q_c(\cdot)$, the set of reproduction values \mathcal{C}_r is never exhaustively listed, as all operations are performed using the product space decomposition.

The coded residual vector can be seen as the “least significant bits”, except that the term “bits” usually refers to scalar quantization. An improved estimation \hat{y} of y is the sum of the approximation vector and the decoded residual vector:

$$\hat{y} = q_c(y) + q_r(r(y)). \quad (9)$$

As shown in Figure 1, this estimator will be used at search time to update the distance estimation between the query x and the database vectors y that are selected as potential neighbors:

$$d(x, y)^2 \approx d_r(x, y)^2 = \|q_c(y) + q_r(r(y)) - x\|^2. \quad (10)$$

The refinement product quantizer q_r is parametrized by its number of subquantizers and the total number of bits. Similar to q_c , we use 8 bits per subquantizer. Therefore the only parameter for the refinement quantizer is the number m' of bytes for each code. The total memory usage per indexed vector is $m + m'$ bytes.

3.2. Algorithm

This subsection details how the refinement codes are used to re-rank the hypotheses provided by the ADC. The resulting approach will be denoted by ADC+R in the experimental section. As for most indexing algorithms, we distinguish between the offline stage and the query stage, during which the system needs to be very efficient. The offline stage consists in learning the indexing parameters and building the indexing structure associated with a vector dataset. It is performed as follows.

1. The quantizers $q_c(\cdot)$ and $q_r(\cdot)$ are learned on a training set.
2. The vector dataset $\mathcal{Y} = \{y_1, \dots, y_n\}$ to be indexed is encoded using q_c , producing codes $q_c(y_i)$ for $i = 1, \dots, n$.

3. The residual vectors are encoded, producing the codes $q_r(y_i - q_c(y_i))$ associated with all the indexed vectors.

Searching a query vector x proceeds as follows:

1. The ADC distance estimation is used to generate a list \mathcal{L} of k' hypotheses. The selected vectors minimize the estimator of Equation 5, which is computed directly in the compressed domain [5].
2. For each vector $y_i \in \mathcal{L}$, the approximate vector \hat{y}_i is explicitly reconstructed using the first approximation $q_c(y_i)$ and the coded residual vector $q_r(y_i)$, see Equation 9. The squared distance estimator $d(x, \hat{y}_i)^2$ is subsequently computed.
3. The vectors of \mathcal{L} associated with the k smallest refined distances are computed.

On output, we obtain a re-ranked list of k approximate nearest neighbors. The choice of the number k' of vectors in the short-list depends on parameters m, m', k' and on the distribution of the vectors. In order for the post-verification scheme to have a negligible complexity, we typically set the ratio k'/k to 2.

3.3. Non exhaustive variant

Up to now, we have only considered the ADC method of [5], which requires an exhaustive scan of the dataset codes. Note however that the re-ranking method proposed in this paper can be applied to any method for which an approximate reconstruction of the indexed vectors is possible, e.g., [11]. In particular, in the experimental section we evaluate our approach with the IVFADC variant of [5], that avoids the aforementioned exhaustive scan by using an inverted file structure. This requires an additional coarse quantizer.

Adapting our re-ranking method to the IVFADC method is straightforward, as this method also provides an explicit approximation of the indexed vectors. In addition to the numbers m and m' of bytes used to encode the vector, this variant requires two additional parameters: the number c of reproduction values of the coarse quantizer and the number v of inverted lists that are visited for a given query. The main advantage of this variant is to compute the distance estimators only for a fraction (in the order of v/c) of the database, at the risk of missing some nearest neighbors if v/c is not large enough. Note finally that the memory usage is increased by $\log_2(c)$ bits (typically 4 bytes), due to the inverted file structure. In the following, the IVFADC method used jointly with our re-ranking method will be denoted by IVFADC+R.

4. EXPERIMENTS

4.1. BIGANN: a billion-sized evaluation dataset

To evaluate ANN search methods, we propose a new evaluation dataset available online: <http://corpus-texmex.irisa.fr>. This benchmark, called BIGANN, consists of 128-dimensional SIFT descriptors (widely adopted image descriptors [8]) extracted from approximately 1 million images. It comprises three distinct subsets:

- base vectors: one billion vectors to search in
- query vectors: 10000 vectors that are submitted to the system
- learning vectors: a set of 100 million vectors to compute the parameters involved in the indexing method.

The groundtruth has been pre-computed: for each query, we provide the k nearest neighbors that are obtained when computing exact Euclidean distance, as well as their square distance to the query

Method	m'	recall@1	@10	@100	time/query
ADC	0	0.075	0.274	0.586	5.626
	8	0.258	0.683	0.951	5.686
	16	0.434	0.895	0.982	5.692
ADC+R	32	0.656	0.970	0.985	5.689
	0	0.088	0.372	0.733	0.074
	8	0.262	0.701	0.962	0.116
IVFADC	16	0.429	0.894	0.982	0.119
	32	0.630	0.977	0.983	0.120
	32	0.630	0.977	0.983	0.120

Table 1. Performance and efficiency measured on 1 billion vectors, $m=8$. The query time is measured in seconds per query. The timings validate the limited impact of the re-ranking stage on efficiency.

vector. The groundtruth for smaller sets ($n=1M, 2M, 5M, 10M, \dots, 200M$ vectors) is also provided. As our own approach does not require many training vectors, we only used the first million vectors from the learning set. All measurements (accuracy and timings) were averaged over the 1000 first queries.

4.2. Evaluation protocol

The search quality is measured by the recall@ r measure, i.e., the proportion of queries whose nearest neighbor is ranked in the first r positions. The curve obtained by varying r corresponds to the distribution function of the ranks, and the point $r=1$ corresponds¹ to the “precision” measure used in [10] to evaluate ANN methods. Also, the recall@ r is the fraction of queries for which the nearest neighbor would be retrieved correctly if a short-list of $k = r$ vectors was verified using exact Euclidean distances.

The efficiency is measured by actual timings.

4.3. Evaluation of the proposed approach

Unless explicitly specified, our approach is evaluated by querying in the whole BIGANN set (i.e., one billion vectors). The performance of the ADC and IVFADC algorithms are given for reference, and compared with the re-ranked versions ADC+R and IVFADC+R. In all experiments, we have set $k=10000$ and $k'=20000$. In addition, for the IVFADC+R we have fixed $c=8192$ and $v=64$, which means that the query is compared to approximately $1/128^{\text{th}}$ of the indexed vectors.

The re-ranking gain: We first consider the improvement brought by the re-ranking stage compared with the reference methods. Figure 2 shows the importance of this re-ranking stage on the recall@ r measure: the performance of PQ+R (resp. IVFPQ+R) is significantly better than that of ADC (resp. IVFADC).

These observations are confirmed by Table 1, which additionally shows that the impact of the re-ranking stage on the query time is limited. As already reported in [5], the IVFADC version is better than the ADC method, at the cost of 4 additional bytes per indexed vector. As expected, this observation remains valid when comparing IVFADC+R with ADC+R.

Performance for a fixed memory usage: Table 2 shows that, for a given memory usage (i.e., for $m + m'$ constant), the re-ranking approach ADC+R achieves similar recall as ADC, at a lower computing cost. The search is approximately two times faster, as the

¹In practice, we are often interested in retrieving the k nearest neighbors ($k > 1$) and not only the nearest neighbor. We do not include these measures in the paper, as qualitative conclusions for $k=1$ remain valid for $k > 1$.

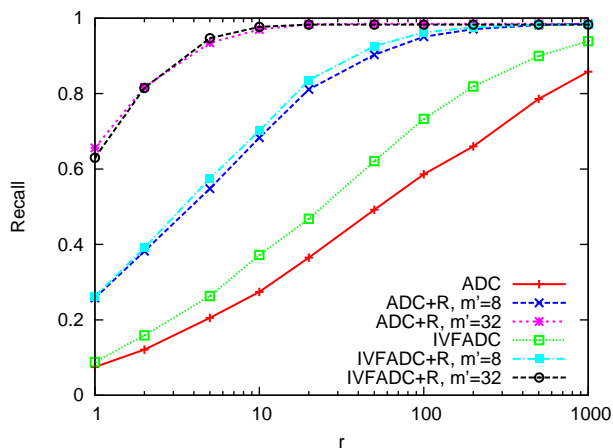


Fig. 2. Searching in one billion vectors: impact of the re-ranking stage on the search accuracy (recall@ r). $m=8$.

bytes/vector	m	m'	recall@1	@10	@100
8	8	0	0.075	0.274	0.586
	4	4	0.056	0.223	0.504
16	16	0	0.245	0.671	0.952
	8	8	0.258	0.683	0.951
32	32	0	0.487	0.956	0.999
	16	16	0.571	0.977	1.000
64	64	0	0.791	1.000	1.000
	32	32	0.832	0.999	1.000

Table 2. Comparative accuracy of ADC ($m'=0$) and ADC+R for the same *total* amount of memory (bytes per indexed vector). In these experiments, the ADC+R approach with $m = m'$ is approximately $2\times$ more efficient than the ADC approach, as the cost of the re-ranking stage is almost negligible, as shown Table 1. These experiments have been performed on the whole BIGANN set.

search time is dominated by the first retrieval stage, whose complexity is asymptotically linear in m and n for large values of n . One can also observe that near perfect neighbors are obtained by increasing the number m' of bytes used to re-rank the queries.

Impact of the dataset size: Figure 3 shows the impact of the database size on the recall@10 measure. The re-ranking stage becomes more important as the database grows in size, due to an increasing number of outliers. Interestingly, the quality of the search degrades more gracefully when using a precise post-verification scheme (with $m'=16$ bytes).

5. CONCLUSION

In this paper, following recent works on multi-dimensional indexing based on source coding, we propose a method to re-rank the vectors with a limited amount of memory, thereby avoiding costly disk accesses. Refining the neighbor hypotheses strongly improves recall at a rather low cost on response time. Our experimental validation is performed on a new public dataset of one billion vectors.

6. ACKNOWLEDGEMENTS

This work was partly realized as part of the Quaero Programme, funded by OSEO, French State agency for innovation.

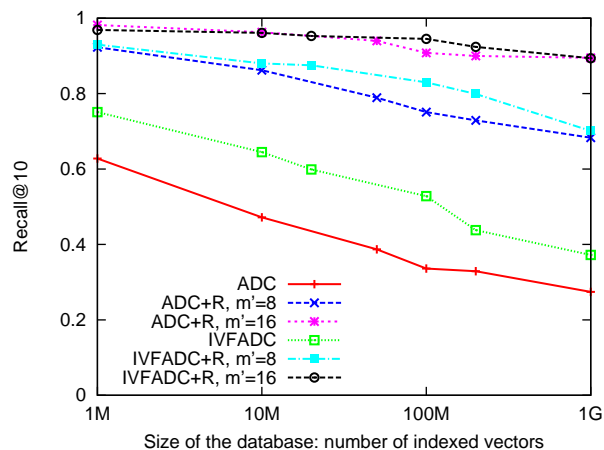


Fig. 3. Impact of the database size on search performance measured by recall@10. $m=8$.

7. REFERENCES

- [1] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *ICDM*, September 2010.
- [2] M. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, April 2008.
- [3] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Symposium on Computational Geometry*, pages 253–262, 2004.
- [4] H. Jégou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *IJCV*, 87(3):316–336, February 2010.
- [5] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1):117–128, January 2011.
- [6] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, June 2010.
- [7] H. Lejsek, F. H. Asmundsson, B. P. Jónsson, and L. Amsaleg. Nv-tree: An efficient disk-based index for approximate search in very large high-dimensional collections. *PAMI*, 31(5):869–883, May 2009.
- [8] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.
- [9] A. Mikulik, M. Perdoch, O. Chum, and J. Matas. Learning a fine vocabulary. In *ECCV*, September 2010.
- [10] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, 2009.
- [11] H. Sandhawalia and H. Jégou. Searching with expectations. In *ICASSP*, Signal Processing. IEEE, March 2010.
- [12] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, chapter 3. MIT Press, March 2006.
- [13] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 7(9):1158–1170, 2000.
- [14] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large databases for recognition. In *CVPR*, June 2008.
- [15] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, November 2008.