Efficient Large-Scale Similarity Search Using Matrix Factorization

Ahmet Iscen, Michael Rabbat, Teddy Furon

▶ **To cite this version:**

Ahmet Iscen, Michael Rabbat, Teddy Furon. Efficient Large-Scale Similarity Search Using Matrix Factorization. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun 2016, Las Vegas, United States. hal-01294736

HAL Id: hal-01294736
https://inria.hal.science/hal-01294736v1

Submitted on 29 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Efficient Large-Scale Similarity Search Using Matrix Factorization

Ahmet Iscen
Inria
Rennes, France
ahmet.iscen@inria.fr

Michael Rabbat
McGill University
Montréal, Canada
michael.rabbat@mcgill.ca

Teddy Furon
Inria
Rennes, France
teddy.furon@inria.fr

Abstract

We consider the image retrieval problem of finding the images in a dataset that are most similar to a query image. Our goal is to reduce the number of vector operations and memory for performing a search without sacrificing accuracy of the returned images. We adopt a group testing formulation and design the decoding architecture using either dictionary learning or eigendecomposition. The latter is a plausible option for small-to-medium sized problems with high-dimensional global image descriptors, whereas dictionary learning is applicable in large-scale scenarios. We evaluate our approach for global descriptors obtained from both SIFT and CNN features. Experiments with standard image search benchmarks, including the Yahoo100M dataset comprising 100 million images, show that our method gives comparable (and sometimes superior) accuracy compared to exhaustive search while requiring only 10% of the vector operations and memory. Moreover, for the same search complexity, our method gives significantly better accuracy compared to approaches based on dimensionality reduction or locality sensitive hashing.

1. Introduction

This paper is about image retrieval and similarity search for large datasets. Image retrieval aims to find the images in a large scale dataset that are most similar to a given query image. Recent approaches [16, 24] aggregate local SIFT [17] features or use deep-learning networks [4] to create a global descriptor vector for each image. Visual similarity is then quantified by measuring the similarity of these vectors (e.g., cosine similarity). If the dataset has N images each represented by a d-dimensional feature vector, then an exhaustive search for each query requires dN operations.

A common approach to accelerate image search is indexing, which operates in sub-linear time [20]. Indexing partitions the feature space $\mathbb{R}^d$ into clusters and computes similarities between the query and dataset vectors that fall in the same or neighboring clusters. Yet, as the dimension d grows, the chance that similar images are assigned to different clusters increases, and the efficiency of these methods collapses [20, 31]. This is problematic in computer vision since most state-of-the-art image descriptors have high intrinsic dimensionality. A recent work tries to solve this by indexing descriptors based on sparse approximation [5].

Another popular approach to efficient image search performs a linear scan over the dataset, computing approximate similarities using compact codes [2, 3, 6, 8, 14, 32]. These techniques have a complexity of $d'N$ where $d' < d$ is the reduced dimensionality of the compact code. The similarity between vectors in $\mathbb{R}^d$ is approximated by the distance between their compact codes. State-of-the-art large scale search algorithms combine indexing strategies with approximated similarities [14].

Recently, a complementary approach inspired by group testing has emerged [11, 27]. Here the goal is to reduce the number of vectors against which the query is compared. The full dataset of N vectors is first summarized by $M \ll N$ group vectors, where each group vector is also d-dimensional. As the name suggests, each group vector represents a small subset of images in the original dataset. These groups are composed by a random partition of the dataset. Computation of the group vectors is performed offline under a specific construction such that a comparison group vector *vs* query vector measures how likely the group contains query matching vectors. Then, when presented with a query, the system compares the query with the group vectors instead of individual image vectors. This reduces the complexity from dN to dM.

Initial attempts [11, 27] considered an *adaptive* group testing approach. M groups are composed from the dataset, and querying proceeds in two stages. In the first stage, the scores between group vectors and the query are computed. They measure how likely their group contains some matching images. Then, in the second stage, the query is compared with individual image vectors for only the mostly likely positive groups. If the groups are roughly balanced in size and the query only matches a small number of group vectors, then the complexity is reduced from dN

to $d(M + N/M)$. Although this results in efficient image retrieval, it has one major drawback: memory usage is increased since the group vectors and mapping from images to groups are stored in addition to the dataset feature vectors. In other words, these works trade complexity for memory. This is not a tractable option for large-N datasets.

In this work, we pursue the idea of deducing which vectors are matching in a database of size N from only $M < N$ measurements. We re-examine the group testing formulation. Rather than a random partition of the dataset into groups followed by a specific construction of the group vectors, we formulate the problem of finding an optimal group testing design for a given image dataset. Removing the restriction to binary designs, the continuous version of this optimization problem turns out to be equivalent to dictionary learning. For small and medium sized datasets, with $N < d$, one can remove the requirement of a sparse design matrix, and then the problem simplifies further to that of a matrix factorization whose solution is given by the SVD.

The paper is organized as follows. Section 2 introduces the problem formulation and notation. Section 3 proposes different techniques to solve the problem depending on the parameters N and d. Section 4 shows the compatibility of our approach with an existing coding method in the literature. Section 5 presents the evaluation of proposed method using real image datasets.

2. Problem statement

The dataset is composed of N d-dimensional vectors $\{\mathbf{x}_i\}_{i=1}^N$ such that $\|\mathbf{x}_i\| = 1$, for all i, and each $\mathbf{x}_i$ is the global feature vector of one image in the dataset. The similarity between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is the scalar product $\mathbf{x}_i^\top \mathbf{x}_j$. Denote by $\mathbf{X}$ the $d \times N$ matrix $[\mathbf{x}_1, \ldots, \mathbf{x}_N]$.

As mentioned in Section 1, we aim to find M group vectors of dimension d, $\{\mathbf{y}_i\}_{i=1}^M$, stored in $d \times M$ matrix $\mathbf{Y}$. Unlike the previous group testing approaches, we do not randomly assign dataset vectors to groups and we do not compute the group vectors according to a specific construction. Our goal is to directly find the best M group vectors globally summarizing the dataset. We call this process the encoding, and we restrict our scope to a linear encoding:

$$\mathbf{Y} = \mathsf{enc}(\mathbf{X}) = \mathbf{X}\mathbf{G}^\top. \tag{1}$$

Given a query image, represented by its global descriptor vector $\mathbf{q}$, we compute the group scores,

$$\mathbf{s} = \mathbf{q}^\top \mathbf{Y}. \tag{2}$$

Finally, we estimate the similarities between query and database vectors $\mathbf{c} = \mathbf{q}^\top \mathbf{X}$ from the measurements $\mathbf{s}$. Again, we assume a linear estimator:

$$\hat{\mathbf{c}} = \mathsf{dec}(\mathbf{s}) = \mathbf{s}\mathbf{H}. \tag{3}$$

Our aim is to design $\mathbf{G} \in \mathbb{R}^{M \times N}$ and $\mathbf{H} \in \mathbb{R}^{M \times N}$ to allow for a fast and accurate search. Note that this setup is similar to the pioneering work of Shi *et al*. [27]: in their paper, $\mathbf{G}$ is indeed a randomly generated binary matrix where $G(i, j) = 1$ if $\mathbf{x}_j$ belongs to the i-th group and $G(i, j) = 0$ otherwise. Hence, in the previous group testing approach, $\mathbf{G}$ captures both how groups are made and how the group vectors are computed (a simple sum in [27]). On the contrary, we look for the best matrix representing the dataset, which will heavily depend on $\mathbf{X}$.

Complexity. Exhaustive search involves computing $\mathbf{q}^\top \mathbf{X}$, which has a complexity of dN. Computing the group measurements (2) takes dM operations, and the decoding (3) takes MN. This gives a complexity of $dM + NM$ for group-testing search, compared to dN operations for exhaustive search. The complexity ratio is thus $\rho = M/N + M/d$, implying that M must be smaller than both N and d to yield efficient queries.

Previous work based on group testing [11, 27] designs groups so that every column of $\mathbf{G}$ has exactly $m \ll M$ ones; i.e., each dataset vector belongs to m groups. This produces a sparse decoding matrix $\mathbf{H}$ which, in turn, yields the better complexity ratio $\rho = M/N + m/d$. However, none of the approaches [11, 27] attempt to optimize $\mathbf{G}$ and $\mathbf{H}$. They either create $\mathbf{G}$ randomly or use a clustering algorithm to coarsely group similar dataset vectors [11]. In the following sections, we discuss two techniques that optimize the matrices $\mathbf{G}$ and $\mathbf{H}$ for a particular dataset $\mathbf{X}$.

We focus on the complexity of performing a query. Determining the optimal encoding and decoding matrices $\mathbf{G}$ and $\mathbf{H}$ requires additional computation applied offline or periodically. We assume that the corresponding complexity is not as critical as in the query stage. Our only requirement is that the complexity of this offline computation be polynomial in N and d to ensure that it is tractable.

3. Proposed solutions

We now provide two alternative solutions for the setup described in Section 2. As we will show in the experimental section, both solutions have advantages and drawbacks, and can be chosen depending on the feature vectors and the number of items in the dataset.

3.1. First solution: Eigendecomposition

In the first approach, we consider finding matrices $\mathbf{G} \in \mathbb{R}^{M \times N}$ and $\mathbf{H} \in \mathbb{R}^{M \times N}$ so that the approximate scores $\hat{\mathbf{c}}$ and exact scores $\mathbf{c}$ are as close as possible. Based on (1), (2) and (3), this amounts to:

$$\begin{aligned} &\underset{\mathbf{G},\mathbf{H}}{\text{minimize}} \quad \sum_{\mathbf{q} \in \mathcal{Q}} \|\mathbf{c} - \hat{\mathbf{c}}\|_2^2 \quad = \\ &\underset{\mathbf{G},\mathbf{H}}{\text{minimize}} \quad \sum_{\mathbf{q} \in \mathcal{Q}} \|\mathbf{q}^T\mathbf{X} - \mathbf{q}^T\mathbf{X}\mathbf{G}^\top\mathbf{H}\|_2^2, \end{aligned}$$

where $\mathcal{Q}$ is assumed to be representative of typical queries. Of course, this distance cannot be zero for all $\mathbf{q} \in \mathbb{R}^d$ since the $N \times N$ matrix $\mathbf{G}^\top\mathbf{H}$ has rank at most $M < N$. We focus on providing accurate scores for typical queries. We use the dataset of vectors itself as a proxy of the typical ensemble of queries. This amounts to replacing $\mathbf{q}$ by $\mathbf{X}$ and to consider the Frobenius matrix norm:

$$\underset{\mathbf{G},\mathbf{H}}{\text{minimize}} \left\|\mathbf{X}^\top\mathbf{X} - \mathbf{X}^\top\mathbf{X}\mathbf{G}^\top\mathbf{H}\right\|_F^2. \tag{4}$$

This problem is commonly solved by eigendecomposition. Let $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$ be the Gramian symmetric matrix associated to $\mathbf{X}$. As a real symmetric matrix, $\mathbf{A}$ is diagonalizable: $\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^\top$, where $\mathbf{U}$ is an orthogonal matrix ($\mathbf{U}^\top\mathbf{U} = \mathbf{U}\mathbf{U}^\top = \mathbf{I}_N$). This means that we can simply assign $\mathbf{G}^\top = \mathbf{U}_M$ and $\mathbf{H} = \mathbf{U}_M^\top$, where $\mathbf{U}_M$ are the eigenvectors associated with the M largest eigenvalues.

In practice, we do not need to compute the Gram matrix $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$. The singular value decomposition (SVD) of $\mathbf{X}$ is defined as $\mathbf{X} = \mathbf{S}\Sigma\mathbf{U}^\top$, where $\mathbf{S}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^\top$, and $\mathbf{U}$ are the eigenvectors of $\mathbf{X}^\top\mathbf{X}$. Hence, this SVD gives us the desired output without having to calculate $\mathbf{A}$. It is worth noting that this solution resembles a well known dimension reduction method: Principal Component Analysis (PCA). However, while PCA is usually employed to reduce the dimensionality of the vectors from d to d' components, in our approach we use it to reduce the number of vectors from N to M. Alternatively, more efficient dimensionality reduction methods, such as sparse projectors [21], can be used to construct $\mathbf{H}$.

The major drawback of this approach is that $\mathbf{H}$ is not sparse. Therefore, the complexity of the decoding (3) is in $\mathcal{O}(MN)$. Hence, this solution is efficient for scenarios where d is larger than N.

3.2. Second solution: Dictionary learning

Dictionary learning has been widely applied in imaging problems, e.g., to obtain efficient representations and discover structure using local patches; see [18] for a survey. Our second solution applies dictionary learning to find a sparse description of the dataset enabling efficient image search. For any query $\mathbf{q}$, we expect the score vector $\mathbf{c}$ to be sparse; the few high-amplitude coefficients correspond to the matching images, and remaining low-amplitude coefficients correspond to non-matching images. Moreover, we do not need the estimate $\hat{\mathbf{c}}$ to be very close to $\mathbf{c}$, per se, as long as the matching images receive a substantially higher score than the non-matching ones.

Because the three steps (1), (2) and (3) of our method are linear, this reconstruction of the similarities through a sparse matrix $\mathbf{H}$ implies a sparse representation of the dataset vectors, which leads to the connection with dictionary learning. Specifically, we aim to approximate $\mathbf{X}$ by $\mathbf{Y}\mathbf{H}$ where $\mathbf{H} \in \mathbb{R}^{M\times N}$ stores the sparse representations of the dataset vectors in terms of columns (so-called atoms) of the dictionary $\mathbf{Y} \in \mathbb{R}^{d\times M}$. This leads to the following optimization problem:

$$\begin{aligned} \underset{\mathbf{Y},\mathbf{H}}{\text{minimize}} \quad & \frac{1}{2}\left\|\mathbf{X} - \mathbf{Y}\mathbf{H}\right\|_F^2 + \lambda\left\|\mathbf{H}\right\|_1 \\ \text{subject to} \quad & \left\|\mathbf{y}_k\right\|_2 \leq 1 \text{ for all } 0 \leq k < M. \end{aligned}$$

The ℓ_1-norm penalty on $\mathbf{H}$ (sum of the magnitude of its elements) encourages a solution where each column of $\mathbf{X}$ can be represented as a sparse combination of columns of the dictionary $\mathbf{Y}$. The level of sparsity depends on λ. Unlike the previous solution of Section 3.1, this scheme is competitive when N is larger than d since we benefit from the reduced complexity of sparse matrix multiplication. An algorithm such as Orthogonal Matching Pursuit (OMP) [7,23] allows us to strictly control the sparsity of $\mathbf{H}$. For a given dictionary $\mathbf{Y}$, OMP finds $\mathbf{H} = [\mathbf{h}_1, \cdots, \mathbf{h}_N]$ by sequentially solving

$$\begin{aligned} \underset{\mathbf{h}_i}{\text{minimize}} \quad & \frac{1}{2}\left\|\mathbf{x}_i - \mathbf{Y}\mathbf{h}_i\right\|_2^2 \\ \text{subject to} \quad & \left\|\mathbf{h}_i\right\|_0 \leq m. \end{aligned}$$

Adopting this algorithm, we control the sparsity of the matrix $\mathbf{H}$ by setting m to a desired value. Note that this solution is directly related with the problem statement in Section 2, even if $\mathbf{G}$ is not directly a part of the solution. The reconstruction of the vectors $\mathbf{X}$ is linear up to an approximation, $\mathbf{X} \approx \mathbf{Y}\mathbf{H}$. Since this is a linear process , we have $\mathbf{Y} = \mathbf{X}\mathbf{G}^\top$ (1) where $\mathbf{G}^\top = \mathbf{H}^+$ (pseudo-inverse). Therefore, the connection is obvious. Furthermore, $\mathbf{G}$ is not needed during the search; what matters is $\mathbf{Y}$ and $\mathbf{H}$.

This solution is similar to the recently proposed indexing strategy based on sparse approximation [5], which also involves training a dictionary $\mathbf{Y}$ and a sparse matrix $\mathbf{H}$. However, the way these matrices are used in [5] is completely different from the approach proposed here. Their framework adheres to a space partitioning approach; it indexes each descriptor in buckets using an inverted file based on the non-zero entries of $\mathbf{H}$. For a given query, their system runs orthogonal matching pursuit (OMP) to find a sparse approximation, and then it calculates distances between the query and the dataset vectors that share the same buckets. In contrast, the method proposed here involves no indexing and makes no direct distance calculations between the query and the dataset vectors. Indeed, this allows us to completely avoid touching dataset vectors at query time.

Similarly, clustering can be used to make groups, as in traditional indexing approaches [20], but the decoding does not perform well for the following reason. The decoding matrix is too sparse: a single non-zero component in each column (this vector belongs to that cluster). This requires

an additional verification step after the decoding step for the vectors in the leading cluster. This is not needed in our method, hence we obtain huge savings in complexity and memory. Our approach can be seen as performing a sort-of soft clustering, where each vector belongs to multiple clusters with different weights.

3.3. Large-scale dictionary learning

When designing an image search system, one must consider large-scale problems consisting of millions to billions of images. As explained in Section 1, our primary goal is an efficient image search system whose query time complexity (computational, and memory) is reduced. Although we have been ignoring the complexity of the encoding phase, by assuming that the complexity of this stage is less critical application-wise, it should remain tractable.

One of the most widely-known dictionary learning algorithms is that proposed by Mairal *et al*. [19]. This algorithm provides a fast implementation and allows other possibilities such as mini-batch learning and online dictionary updates. These features make it an attractive algorithm for large-scale problems. However, the training time increases dramatically with M for large-N datasets, as reported in Section 5. Even though this calculation needs to be done only once in the offline stage, we still need a scalable training approach to index all dataset vectors easily.

One solution is to use a subset of dataset vectors as a surrogate for the entire dataset. Once the dictionary $\mathbf{Y}$ is trained on the subset, a less expensive sparse decoding algorithm, such as OMP, can be used to compute the matrix $\mathbf{H}$ for the entire dataset.

Elhamifar *et al*. [9] propose a solution similar to dictionary learning, with the sole aim of finding representatives from the data. A related approach is to use *coresets* [1]. A coreset $\mathbf{C}$ is a problem-dependent approximation of a dataset $\mathbf{X}$. Feldman *et al*. [10] show that for every $\mathbf{X}$ and $\epsilon > 0$ there exists a coreset $\mathbf{C} \in \mathbb{R}^{d \times N'}$, $N' < N$, for which the following inequality holds:

$$(1-\epsilon)\cdot \min_{\mathbf{H}\in\mathbb{R}^{M\times N}} \|\mathbf{X}-\mathbf{Y}\mathbf{H}\|_F^2 \leq \min_{\widetilde{\mathbf{H}}\in\mathbb{R}^{M\times N'}} \left\|\mathbf{C}-\mathbf{Y}\widetilde{\mathbf{H}}\right\|_F^2 \leq (1+\epsilon)\cdot \min_{\mathbf{H}\in\mathbb{R}^{M\times N}} \|\mathbf{X}-\mathbf{Y}\mathbf{H}\|_F^2 .$$

Typically, $\mathbf{C}$ has many fewer columns than $\mathbf{X}$, thereby summarizing the whole dataset with just a few representatives. The main advantage of this approach is its speed. Finding a coreset for a large-scale dataset takes a short time, only a few seconds in our experiments. Then, running dictionary learning on the coreset is significantly faster than on the original dataset. We empirically evaluate the speedup and the effect on accuracy in the experimental section.

4. Compressed dictionaries

Instead of dealing with a database of N image vectors of length d, our novel approach now manages a database of M group vectors of the same dimension. Compared to a linear scan, we reduce the number of comparisons from N to M, and yet, rank N items based on their estimated score.

Nevertheless, our scheme remains compatible with the traditional coding methods briefly introduced in the introduction. Instead of a linear scan browsing group vectors, we can add on top of our method an approximate search. This can take the form of either an embedding producing compact representations of the group vectors, or an indexing structure finding the closest group vectors w.r.t. a query. This improves even further the overall efficiency.

Case study: Combination with PQ-codes. An embedding offers a compact representation of group vectors allowing a fast approximation of their dot products with the query. PQ-codes [14], for instance, are *a priori* not compliant since they operate on Euclidean distances. We convert Euclidean distance to cosine similarity in the following way. Each group vector $\mathbf{y}$ is split into ℓ subvectors $\tilde{\mathbf{y}}_u$, where $1 \leq u \leq \ell$. Each subvector $\tilde{\mathbf{y}}_u$ is quantized using the codebook $\mathcal{C}_u = \{\mathbf{c}_{i,u}\}_{i=1}^{Q}$: $v_u = \arg\min_{1\leq i\leq Q} \|\tilde{\mathbf{y}}_u - \mathbf{c}_{i,u}\|$. The compact representation of $\mathbf{y}$ is the list of codeword indices $(v_1,\ldots,v_\ell) \in \{1,\ldots,Q\}^\ell$. This is exactly the same encoding stage as the original PQ-codes [14].

The dot product query *vs* group vector is approximated by the dot product query *vs* quantized group vector:

$$\mathbf{q}^\top \mathbf{y} = \sum_{u=1}^{\ell} \tilde{\mathbf{q}}_u^\top \tilde{\mathbf{y}}_u \approx \sum_{u=1}^{\ell} \tilde{\mathbf{q}}_u^\top \mathbf{c}_{v_u,u}, \qquad (5)$$

where $\tilde{\mathbf{q}}_u$ is the u-th subvector of the query. As in the original application of PQ-codes, the quantities $\{\tilde{\mathbf{q}}_u^\top \mathbf{c}_{i,u}\}$ are computed at query time and stored in a lookup table for evaluating (5) efficiently over a large number of group vectors. Using approximate dot products is an additional source of error, but experiments in the next section show that the decoding schemes described above gracefully handle this.

5. Experiments

After detailing the experimental protocol, we report retrieval performance results together with a comparison with other image retrieval approaches.

5.1. Experimental setup

Datasets. We evaluate our retrieval system using the Oxford5k [25] and Paris6k [26] datasets, which contain 5,063 and 6,412 images respectively. For large-scale experiments we add 100,000 Flickr distractor images [25], resulting in datasets referred as to Oxford105k and Paris106k. Additionally, we use the Yahoo Flickr Creative Commons 100M

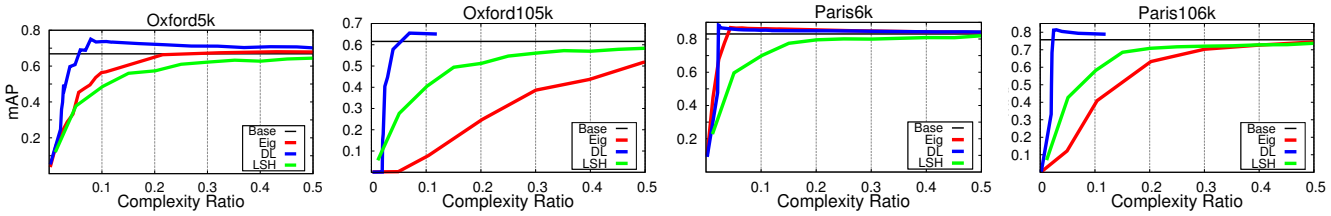

Figure 1. Comparison of eigendecomposition, dictionary learning (DL), and LSH [6]. DL gives better performance, all the more so as the dataset is large. We only evaluate DL up to $M/N = 1/10$ for Oxford105k and Paris106k. Performance eventually converges to the baseline after this point.

dataset [29] (referred as to Yahoo100M), which comprises about 100 million image vectors. For comparison with other works, we also run experiments on the Holidays [13] and UKB [22] datasets.

For each dataset, we follow its standard procedure to evaluate performances. The *mean Average Precision* (mAP) measures the retrieval quality in all datasets except for UKB, where the performance is gauged by $4\times$recall@4.

Features. For most of our experiments, we use the state-of-the-art R-MAC features [30]. Depending on the network used, these features have dimensionality of either $d = 512$ or $d = 256$.[1] In section 5.3, we use T-embedding features [16] with $d = 8,064$ to allow a more direct comparison with the most similar concurrent methods. For Yahoo100M, we use VLAD [15] with $d = 1,024$ as in [28].

Complexity analysis. We report the complexity ratio, $\rho = (Md + s)/dN$, where $s = \text{nnz}(\mathbf{H})$ is the number of non-zero elements of matrix $\mathbf{H}$. For the eigendecomposition, we set $s = MN$, whereas for dictionary learning (Section 3.2), m controls the sparsity of $\mathbf{H}$ making the complexity ratio $\rho = M/N + m/d$. Unless otherwise specified, we set $m = 10$ for R-MAC features; when $d = 512$ then decoding contributes only 0.02 to ρ (i.e., 2% of the complexity of exhaustive search). The memory ratio, the ratio of the memory required compared to that of exhaustive search, is equal to ρ for non-sparse $\mathbf{H}$. When $\mathbf{H}$ is sparse, we need to store mN scalars and their indices, making the memory ratio $M/N + m/d + m\log_2(M)/d \approx \rho$.

5.2. Retrieval performance

We first evaluate our system for different M using either eigendecomposition or dictionary learning solutions. We also include the popular sketching technique LSH [6], which approximates similarity by comparing binary compact codes of length $d' = \rho d$. We measure the retrieval performance in terms of mAP and complexity ratio as mentioned in Section 5.1.

Figure 1 shows the retrieval performance for different complexity ratios. It is clearly seen that eigendecomposition suffers at low complexity ratio in large-scale datasets. This is expected because we must set M to a very small value to obtain a low complexity ratio since the decoding matrix $\mathbf{H}$ is not sparse in this solution. On the other hand, we can set M to a much higher value for a given complexity ratio using dictionary learning since $\mathbf{H}$ is sparse.

Our variant based on dictionary learning performs better than the baseline on all datasets. One would expect the performance to be worse than baseline for $M \ll N$ due to loss of information, but this is surprisingly not the case. A possible explanation is that dictionary learning "denoises" similarities between vectors. In computer vision, each image is represented by a global vector, which is usually obtained by aggregating local features, such as SIFT, or response maps from convolutional neural networks (in the case of R-MAC). These local features are obtained from both useful structure of the scene and also from clutter.

Our interpretation is that dictionary learning decreases the impact of features extracted from clutter patches because they are not common across the image collection; i.e., it favors the frequent visual patterns in the image collection. To explore this phenomenon further, we plot the distribution of matching and non-matching vector similarities from Oxford5k using the original global descriptors. We repeat the same process using the reconstructed similarities from dictionary learning. As we see in Figure 2, both reconstructed similarity distributions have a lower variance than the original distributions. This is especially true for the non-matching distribution. This variance reduction increases the separation between the distributions, which translates to the better performance of our dictionary learning method.

Sparsity of H is controlled by parameter m in dictionary learning (see Section 3.2). This is an important factor in the complexity ratio ρ. The ratio between m and d contributes to ρ independently from M. It is possible to set this ratio to a small value to eliminate its influence.

We compute a dictionary of M atoms and we calculate several matrices $\mathbf{H}$ by applying OMP with different m. We plot the retrieval performance for different m and M in Figure 3. In most cases, the performance does not vary much w.r.t. m. The biggest difference is observed for Oxford105k where larger m leads to better performance for small M.

The dimensionality of the vectors is an important factor affecting the overall complexity. Lower dimensionality im-

[1] Features available online: `ftp://ftp.irisa.fr/local/texmex/corpus/memvec/cvpr16/rmac/`

	Original	Dict. Learning
Pos. μ	0.29	0.28
Pos. σ	0.16	0.15
Neg. μ	0.02	0.02
Neg. σ	0.06	0.04

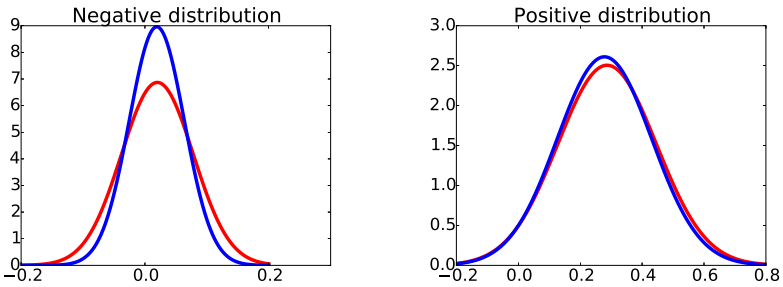

Figure 2. Distributions of matching and non-matching vector similarities from Oxford5k dataset. Red (blue) curves represent distributions of true (resp. reconstructed) similarities. The main improvement comes from the reduction of variance under the negative distribution.

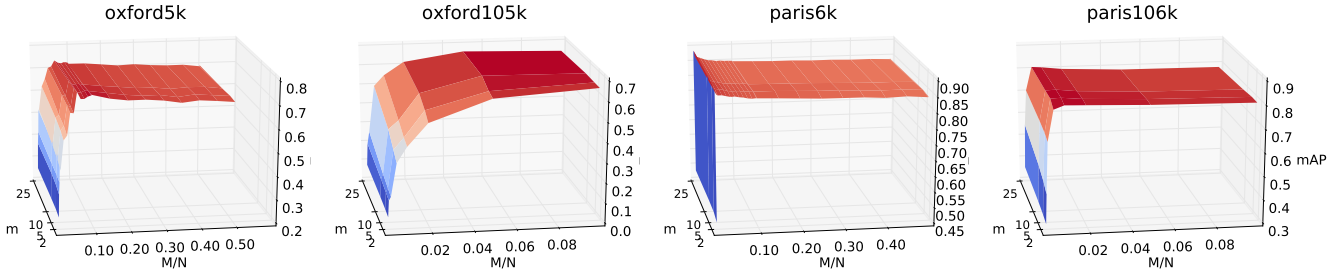

Figure 3. Retrieval performance with different M and m. Varying m does not affect the performance in most cases, except for Oxford105k, where increasing m improves performance for small M.

plies lower complexity and less memory usage. Although our experiments up to now are done in what can be considered as a low-dimensional feature space ($d = 512$), we evaluate our system with even smaller features, $d = 256$, in Figure 4. The results are similar to those for $d = 512$, although the accuracy of eigendecomposition increases at a slower rate for large N.

The training stage computes $\mathbf{Y}$ and $\mathbf{H}$ and is performed only once and offline. However, it is important that this stage is scalable for updating the dictionary if needed. Experimentally, a small number of iterations (≈ 100) is sufficient for dictionary learning. This does not require much training time. Using Mairal *et al.* 's algorithm [19], we report the duration of the offline training on Figure 5. All experiments are done on a server with Intel® Xeon® E5-2650 2.00GHz CPU and 32 cores. The training time is reasonable for all datasets, but it increases dramatically with M in large datasets. Other training procedures would be necessary for handling large M and N.

Coresets, as explained in Section 3.3, reduce the training time even further for large datasets. Instead of using the entire dataset, we find a coreset $\mathbf{C}$ which represents the data with a few representatives vectors to train the dictionaries. We report results for coresets of different sizes in Table 1. Empirically, we achieve a similar performance by training on coresets of vectors. This allows us to train the dictionary for larger M in just a few minutes. Note that Paris106k has fast training time even without coresets. This is because the best performance for this dataset is obtained with $M = 532$, a rather small value. The drawback to using coresets is that $\mathbf{H}$ is less sparse: $m = 50$. This results in the same performance but slightly higher complexity.

	Oxford105k		Paris106k	
	mAP	**Time**	**mAP**	**Time**
$\lvert\mathbf{C}\rvert = N/10$	60.1±1.1	14.6	78.3±1.0	1.8
$\lvert\mathbf{C}\rvert = N/5$	62.1±1.2	16.9	79.2±0.8	2.3
$\lvert\mathbf{C}\rvert = N/2$	62.7±0.4	23.9	79.5±0.4	3.3
$\mathbf{X}$	65.5	45.5	81.2	5.3

Table 1. Performance and training time (in minutes) using coresets to train the dictionary. M is set to $5,257$ and 532 for Oxford105k and Paris105k respectively, and $m = 50$. Each experiment is run 5 times, and we report the mean and the standard deviation.

The search time is the average number of seconds to respond to a query. Although comparing vector operations is reliable in general, we also include the actual timings. Exhaustive search takes $0.029s$ on Oxford105k and $0.03s$ on Paris106k (average per query). Our method takes $0.003s$ on Oxford105k ($M = 5,257$), and $0.001s$ on Paris106k ($M = 532$), with higher mAP than exhaustive search.

5.3. Comparison with other methods

We compare our system with other image retrieval approaches. First we compare with the popular FLANN toolbox [20] using Oxford105k and R-MAC features. We set the target precision to 0.95 and use the "autotuned" setting of FLANN, which optimizes the indexing structure based on the data. We repeat this experiment 5 times. The average speed-up ratio provided by the algorithm is 1.05, which corresponds to a complexity ratio of 0.95. In other words, FLANN is ineffective for these R-MAC descriptors, most

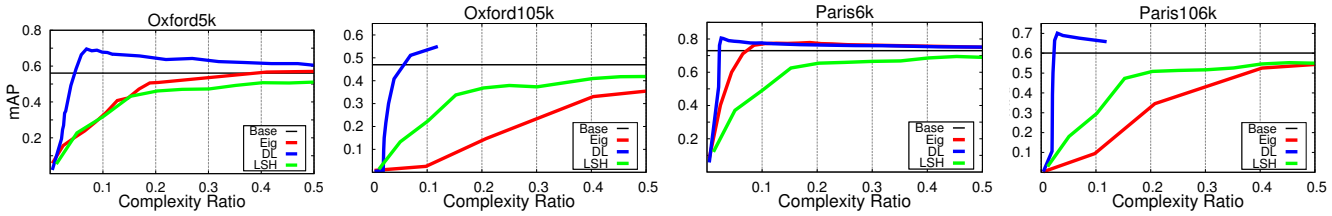

Figure 4. Retrieval performance using smaller features: $d = 256$.

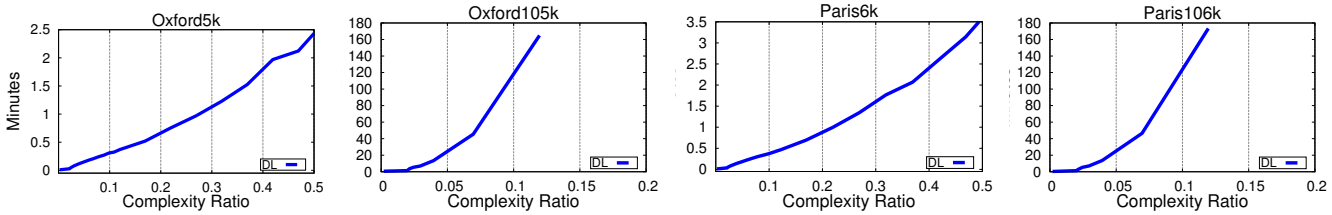

Figure 5. Offline training time needed for dictionary learning with 100 iterations.

	Mem. Ratio	Holidays	Oxford5k	UKB
Exhaustive	1.0	77.1	67.4	3.63
Iscen *et al.* [11]-Kmeans	1.4	**76.9**	67.3	**3.63**
Iscen *et al.* [11]-Rand	1.4	75.8	62.0	**3.63**
Shi *et al.* [27] w/ bp.	1.4	75.5	64.4	**3.63**
Borges *et al.* [5]	1.0	59.2	59.9	3.43
LSH [6]	0.4	73.9	65.8	3.61
PCA	0.4	75.4	64.3	3.61
Shi *et al.* [27] w/o bp.	0.4	8.7	24.1	1.33
Ours - Eigen.	0.4	**76.9**	67.7	**3.63**
Ours - Dict. Learn.	0.4	55.2	**68.8**	3.59

Table 2. Comparison in image retrieval for a given complexity ratio of 0.4. This experiment uses long t-embedding features ($d = 8,096$). Eigendecomposition and dictionary learning generally performs better at lower memory ratio.

	Mem. Ratio	Oxf5k	Oxf105k	Paris6k	Paris106k
Exhaustive	1.0	66.9	61.6	83.0	75.7
[11]-Kmeans	1.1	65.6	61.2	79.7	75.7
[11]-Rand	1.1	25.1	43.7	21.2	44.4
[27] w/ bp.	1.1	15.4	28.1	18.7	37.7
[5]	1.0	8.5	22.7	8.2	18.9
LSH [6]	0.1	48.6	40.5	70.1	58.2
PCA	0.1	58.1	8.0	86.1	38.9
Ours-Eigen.	0.1	56.8	8.0	**86.3**	40.9
Ours-D.L.	0.1	**73.7**	**65.5**	85.3	**78.9**

Table 3. Comparison with R-MAC features ($d = 512$) and 0.1 complexity ratio.

likely due to their high intrinsic dimensionality ($d = 512$): as discussed by its authors [20], FLANN is not better than exhaustive search when applied to truly high-dimensional vectors. In contrast, our approach does not partition the feature space and does not suffer as much the curse of dimensionality. Our descriptors are whitened for better performance [12], which tends to reduce the effectiveness of partitioning-based approaches.

Next we compare our method with other group testing and indexing methods in the image retrieval literature. To have a fair comparison, we report the performance using the same high-dimensional features ($d = 8,064$), same datasets, and the same complexity ratio as the group testing methods. Additionally, we also compare our scores to a dictionary learning-based hashing method [5], LSH [6] and PCA, where dimensionality of vectors is reduced such that $d' = 0.4d$. Table 2 shows the comparison for a fixed complexity ratio. We outline two observations. First, eigendecomposition works well in these experiments. This is especially true for the Holidays dataset where $N = 1,491$ and $d = 8,064$; large M can be used while keeping the complexity ratio low since $N < d$. This is clearly a scenario where it is plausible to use the eigendecomposition approach. Second, dictionary learning performs poorly for Holidays. This dataset contains only $1,491$ images, which constrains the size of the dictionary M to be small and prevents sparsity: the best parameters (via cross-validation) are found to be $M = 519$ and $m = 409$, giving $\rho = 0.4$. Note that this experiment uses long t-embedding descriptors ($d = 8,096$) in small and mid-scale datasets. Most likely, these features have low intrinsic dimensionality, and PCA and LSH are thus favored. Table 3 uses shorter R-MAC features ($d = 512$) for comparison. The increase in performance is more significant, especially for large datasets.

Yahoo100M is a recently released large-scale dataset consisting of approximately 100M images. Since there is no manually annotated ground-truth, we use the following evaluation protocol: a dataset vector is considered to match the query if its cosine similarity is at least 0.5. There are 112 queries randomly selected from the dataset. Each query has between 2 and 96 matches, and 11.4 matches on average. Table 5.2 shows visual examples of queries and matches.

This dataset is split into chunks of $N' = 100$k images. We run dictionary learning and OMP independently to learn matrices $\mathbf{Y}$ and $\mathbf{H}$ for each chunk, setting $M' = N'/100$ and $m = 100$. Overall, it results in $M = N/100$. We

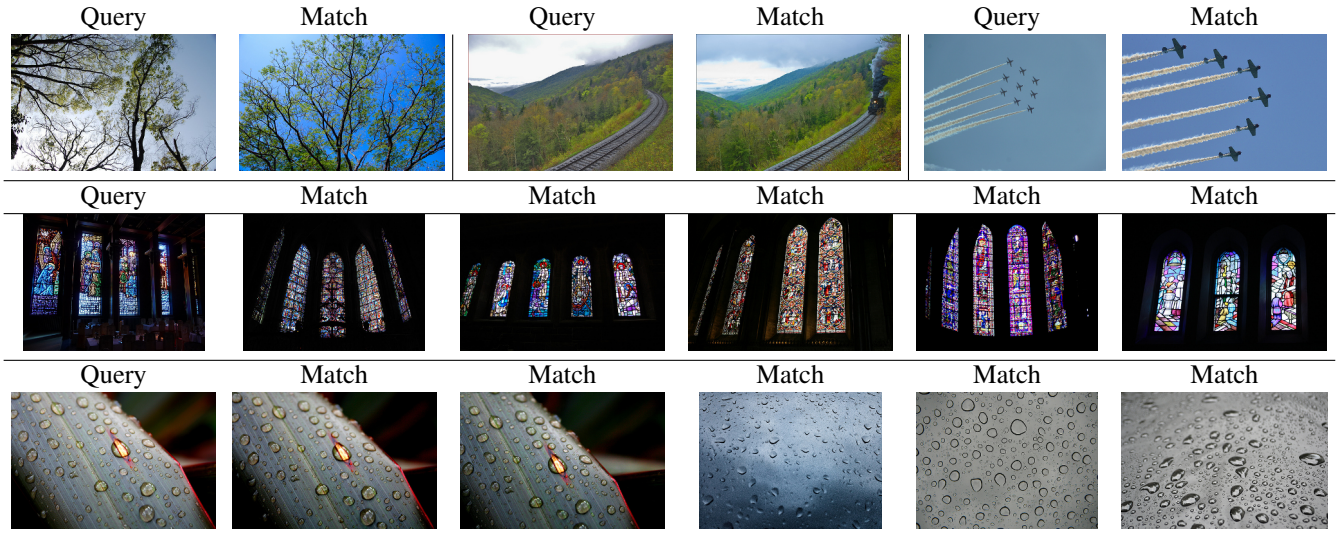

Table 4. Some examples of match and query in Yahoo100M dataset. Two vectors are considered a match if their similarity is above 0.5.

	$M = N/200$		$M = N/100$		$M = N/50$	
	mAP	ρ	**mAP**	ρ	**mAP**	ρ
$m = 100$	85.7	0.105	89.4	0.11	92.8	0.12
$m = 50$	81.0	0.055	84.7	0.06	87.4	0.07
$m = 20$	61.8	0.025	71.4	0.03	78.2	0.04

Table 5. Performance (mAP) and complexity ratio (ρ) in Yahoo100M for different M and m.

	Baseline	Our Method	$b = 8$	$b = 64$
Oxford5k	66.9	73.4	73.1	72.9
Paris6k	83.0	88.1	87.7	85.6
Oxford105k	61.6	65.5	63.1	30.4
Paris106k	75.7	81.2	80.9	76.8

Table 6. Combination of our method with PQ-codes. We use $M = 350$ for Oxford5k, $M = 30$ for Paris6k, $M = 5257$ for Oxford105k, and $M = 532$ for Paris106k.

can perform this offline stage in parallel. At query time, we pool scores from each chunk together and sort them to determine a final ranking. When we evaluate the retrieval performance, we obtain a mAP of 89.4 with $\rho \approx 1/10$. This is a significant increase compared to running the same setup with LSH, which results in a mAP of 70.9. Furthermore, it is still possible for the dictionary learning approach to obtain very good performance with $\rho < 1/10$ by setting M and m to smaller values as shown in Table 5.3.

Similar to other datasets, we apply coresets for the Yahoo100M dataset. We learn a coreset for each chunk separately, which makes its calculation feasible. We set $|\mathbf{C}| = N/2$, $m = 100$ and $M = N/100$, and obtain a mAP of 87.9 compared to a mAP of 89.4 using the entire chunks.

Compatibility with coding methods. One of the main strengths of our method is its complementarity with other popular coding strategies in computer vision. We combine our method with PQ-codes [14] as explained in Section 4. We use $\ell = d/b$ subvectors for different values of b and $Q = 256$ codewords per each subquantizer (except for Paris6k where $Q = 16$ due to small M). This reduces the term $\mathcal{O}(M \times d)$ by a factor of b if we neglect the fixed cost of complexity of building the lookup table.

Table 6 shows the difference of performance with and without PQ-codes. Observe that the performance remains almost the same for $b = 8$. The compression factor by PQ-code is significant (8 floats replaced by 1 byte).

6. Conclusion

This paper lowers the complexity of image search by reducing the number of vector comparisons. We formulate the image search problem as a matrix factorization problem, which can be solved using eigendecomposition or dictionary learning. We show that the former is a plausible option for small datasets, whereas the latter can be applied for large-scale problems in general. When applied to real datasets comprising up to 10^8 images, our framework achieves a comparable, and sometimes better performance, than exhaustive search within a fraction of complexity. It is worth noting that this approach is complementary to other indexing/approximated similarity approaches such that it can be combined to further increase efficiency.

Acknowledgments

This work was supported, in part, by the Natural Sciences and Engineering Research Council of Canada through grant RGPAS 429296-12, and by the French National Project IDFRAud (ANR-14-CE28-0012). Portions of this work were completed while the first author was visiting McGill University.

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.

[2] R. Arandjelović and A. Zisserman. Extremely low bit-rate nearest neighbor search using a set compression tree. *IEEE Trans. PAMI*, 2014.

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

[4] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. In *ECCV*, 2014.

[5] P. Borges, A. Mourão, and J. Magalhães. High-dimensional indexing by sparse approximation. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, 2015.

[6] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, May 2002.

[7] G. M. Davis, S. G. Mallat, and Z. Zhang. Adaptive time-frequency decompositions with matching pursuit. In *SPIE's International Symposium on Optical Engineering and Photonics in Aerospace Sensing*, pages 402–413, 1994.

[8] W. Dong, M. Charikar, and K. Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *SIGIR*, pages 123–130, July 2008.

[9] E. Elhamifar, G. Sapiro, and R. Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *CVPR*, 2012.

[10] D. Feldman, M. Feigin, and N. Sochen. Learning big (image) data via coresets for dictionaries. *Journal of Mathematical Imaging and Vision*, 46(3):276–291, 2013.

[11] A. Iscen, T. Furon, V. Gripon, M. Rabbat, and H. Jégou. Memory vectors for similarity search in high-dimensional spaces. *arXiv preprint arXiv:1412.3328*, 2014.

[12] H. Jégou and O. Chum. Negative evidences and cooccurrences in image retrieval: The benefit of PCA and whitening. In *ECCV*, October 2012.

[13] H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, October 2008.

[14] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. PAMI*, 33(1):117–128, January 2011.

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

[16] H. Jégou and A. Zisserman. Triangulation embedding and democratic kernels for image search. In *CVPR*, June 2014.

[17] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[18] J. Mairal, F. Bach, and J. Ponce. Sparse modeling for image and vision processing. *arXiv preprint arXiv:1411.3230*, 2014.

[19] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11:19–60, 2010.

[20] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. PAMI*, 36, 2014.

[21] R. Negrel, D. Picard, and P.-H. Gosselin. Dimensionality reduction of visual features using sparse projectors for content-based image retrieval. In *ICIP 2014*, pages 2192–2196, 2014.

[22] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, June 2006.

[23] Y. C. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *ASILOMAR*, pages 40–44, 1993.

[24] F. Perronnin and C. R. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, June 2007.

[25] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, June 2007.

[26] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, June 2008.

[27] M. Shi, T. Furon, and H. Jégou. A group testing framework for similarity search in high-dimensional spaces. In *ACM Multimedia*, November 2014.

[28] E. Spyromitros-Xioufis, S. Papadopoulos, I. Kompatsiaris, G. Tsoumakas, and I. Vlahavas. A comprehensive study over VLAD and product quantization in large-scale image retrieval. *IEEE Trans. on Multimedia*, 2014.

[29] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. Yfcc100m: The new data in multimedia research. *Commun. ACM*, 59(2), 2016.

[30] G. Tolias, R. Sicre, and H. Jégou. Particular object retrieval with integral max-pooling of cnn activations. *ICLR*, 2016.

[31] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

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