Large-Scale Instance-Level Image Retrieval

Giuseppe Amato, Fabio Carrara*, Fabrizio Falchi, Claudio Gennaro, Lucia Vadicamo

Institute of Information Science and Technologies, Italian National Research Council, 56124 Pisa, Italy

Abstract

The great success of visual features learned from deep neural networks has led to a significant effort to develop efficient and scalable technologies for image retrieval. Nevertheless, its usage in large-scale Web applications of content-based retrieval is still challenged by their high dimensionality. To overcome this issue, some image retrieval systems employ the product quantization method to learn a large-scale visual dictionary from a training set of global neural network features. These approaches are implemented in main memory, preventing their usage in big-data applications. The contribution of the work is mainly devoted to investigating some approaches to transform neural network features into text forms suitable for being indexed by a standard full-text retrieval engine such as Elasticsearch. The basic idea of our approaches relies on a transformation of neural network features with the twofold aim of promoting the sparsity without the need of unsupervised pre-training. We validate our approach on a recent convolutional neural network feature, namely Regional Maximum Activations of Convolutions (R-MAC), which is a state-of-art descriptor for image retrieval. Its effectiveness has been proved through several instance-level retrieval benchmarks. An extensive experimental evaluation conducted on the standard benchmarks shows the effectiveness and efficiency of the proposed approach and how

*Corresponding author.

Email addresses: giuseppe.amato@isti.cnr.it (Giuseppe Amato), fabio.carrara@isti.cnr.it (Fabio Carrara), fabrizio.falchi@isti.cnr.it (Fabrizio Falchi), claudio.gennaro@isti.cnr.it (Claudio Gennaro), lucia.vadicamo@isti.cnr.it (Lucia Vadicamo)

Preprint submitted to Journal of Information Processing and Management September 3, 2019

it compares to state-of-the-art main-memory indexes.

Keywords: image retrieval, deep features, surrogate text representation, inverted index

1. Introduction

Full-text search engines on the Web have achieved great results in terms of efficiency thanks to the use of inverted index technology [1, 2, 3]. In the last years, we experienced an increasing interest of the research community in the case of retrieval of other forms of expression, such as images [4, 5]; nevertheless, the development in those cases was not as rapid as text-based paradigms. In the field of image retrieval, this was initially due in part to the ineffectiveness of hand-crafted features used by instance-level and content-based image retrieval. However, since 2014 we have had a great development of new learned features obtained by training neural networks – in particular convolutional neural networks (CNN). Differently from text, in which inverted indexes perfectly marry the sparse document representation in standard vector models, learned image descriptors tend to be dense and compact, thus making directly unfeasible the usage of mature text-tailored index technologies. While efficient index structures for this type of data exist [6, 7, 8], they usually come with caveats that prevent their usage in very large-scale scenarios, such as main-memory-only implementations and computationally expensive indexing or codebook-learning phases.

In a nutshell, the idea is to use image representations extracted from a CNN, often referred to as *Deep Features*, and to transform them into text so that they can be indexed with a standard text search engine.

The application focus of this work is on a scenario of image retrieval in a large-scale context, with an eye to scalability. This aspect is often overlooked by the literature, most of the image retrieval systems are designed to work in main memory and many of these cannot be distributed across a cluster of nodes [9]. Many techniques present in literature try to tackle this problem by heavily

compressing the representation of visual features to adapt more and more data to the secondary memory. However, these approaches to indexing are not able to scale because sooner or later response times become unacceptable as the size of the data to be managed increases.

In particular, our general approach is based on the transformation of deep features, which are (dense) vectors of real numbers, into sparse vectors of integer numbers. The transformation in integers is necessary to deal with textual representations of the vectors, as it will be explained better below, in which the components of the vectors are in fact translated by "term frequency" of these textual representations. Sparseness is necessary to achieve sufficient levels of efficiency exactly as it does for search engines for text documents. To obtain this two-fold result, we will analyze two approaches: one based on permutations and one based on scalar quantization.

The present paper is the evolution of previous works [10, 11, 12, 13, 14]. In [10] the idea of representing metric objects as permutations of reference objects to construct an inverted index that In [11], and allowing us to take advantage of a standard text search engine without having to implement the inverted index. In [12], the authors introduced the idea of Deep Permutations that applies to the deep feature vectors and in which the components of the vectors themselves are permuted. In [13] and [14] an extension of the technique of Deep Permutations is presented, in the former using the surrogate text representation and R-MAC, and in the latter taking into account the negative components of R-MAC. In [14], we have also proved that this general approach can be implemented on top of Elasticsearch by showing how such a retrieval system is able to scale to multiple nodes. In the earlier attempt [15], we have presented a preliminary draft of quantization approach on deep features extracted from the Hybrid CNN[1], which is less effective but has the advantage of being partly spare.

The original contribution of the present work consists in introducing a new approach of surrogate representation for deep features based on *Scalar Quan-*

[1] http://github.com/BVLC/caffe/wiki/Model-Zoo.

tization. We present this approach in a unified framework of representation of deep features using surrogate text together with the technique based on Deep Permutations, and we compare it with the scalar quantization technique. We have also extended the experimental evaluation by adding two more benchmarks and, regarding efficiency, we also considered the size of the indexes as well as their percentage of use.

The rest of the paper is organized as follows: Section 2 surveys the relevant related work. Section 3 provides a brief background about the Deep Features. In Section 4, the main contribution of this paper, namely the Surrogate Text Representation is presented. Section 5 shows experimental results, and finally Section 6 gives concluding remarks. Table 1 summarizes the notation used throughout this manuscript.

2. Related Work

To frame our work in the context of scientific literature, we refer to the survey of Zheng et al. [16], which organizes the literature according to the codebook size, i.e. large/medium-sized/small codebooks. Although this organization, according to the authors, is relevant to local features (which were defined as "sift-based" by the authors), we think it can be extended to deep features and representation-focused neural models in general [17]. We think our work belongs to medium-sized (or even large-sized codebooks), which rely on their sparsity to exploit inverted indexes and the trade-off between accuracy and efficiency is a major influencing factor.

In this respect, the scientific literature devoted to mitigating the complexity of computing the matching between local features and based on the generation of visual words, i.e. Bag-of-Words (BoW), are very close to our work in the spirit [18]. If we limit ourselves to consider only the works that exploit the sparsity of the BoW model (originated from a work by Sivic et al. [19]) paper [20] uses an inverted index on secondary memory to implement a scalable visual object-retrieval system based on the SIFT descriptor. We refer the reader

to the survey [16] for more information about other approaches that quantize local features into visual words. In the following discussion, we concentrate our attention on techniques that try to deal with deep features with an inverted index.

A permutation-like approach, which relies on the use of special anchor objects called pivots, were used in PPP-Codes index [21] to index a collection of 20 million images processed by a deep convolutional neural network. However, the proposed approach relies on an index specifically developed to manage permutations in secondary memory called *recursive Voronoi partitioning*.

Liu et al. [6] proposed a framework that adapts the BoW model and inverted table to deep feature indexing, which is similar to the one we propose. However, for large-scale datasets, Liu et al. have to build a large visual dictionary that employs the product quantization method to learn it from a training set of global deep features. This approach also uses a specialized index that combines inverted table and hashing codes. Moreover, it has been tested only with sparse outdated deep features that exhibited low accuracy.

Some other works try to treat the features in a convolutional layer as local features [22, 23]. This way, a single forward pass of the entire image through the CNN is enough to obtain the activations of its local patches, which are then encoded using *Vector of Locally Aggregated Descriptors* (VLAD). A similar approach uses the BoW encoding instead of VLAD to take advantage of sparse representations for fast retrieval in large-scale databases. However, although authors claim that their approach is very scalable in terms of search time, they did not report any efficiency measurements and experiments have been carried out on datasets of limited size.

In [7], the authors quantized the deep features coming from a VGG16 pre-trained network with a codebook of size 25k and employed an inverted index for efficiency. Also in this case, although authors claim that their BoW-based representation is highly sparse, allowing for fast retrieval using inverted indices, no experimental evidence was offered.

An approximate nearest neighbor algorithms based on product quantization

(PQ), which exploits an inverted index is presented in [24]. In PQ, the original vector is divided into M sub-vectors which are then independently quantized. A codebook is learned via k-means for each of the M sub-division, and each sub-vector is compressed by storing the nearest centroid index. In particular, we focus on PQ-compressed inverted indexes, denoted IVFPQ, which are implemented in the open source FAISS library [8]. In IVFPQ, the feature space is partitioned into N Voronoi cells, each associated with a particular posting list of the inverted file. Each posting list contains the PQ-compressed difference between samples belonging to that cell and its centroid. When building the index, both the cell centroids and the PQ-compression codebooks have to be pretrained on a subset of the data. When querying the index, FAISS probes the posting lists of the P Voronoi cells nearest to the query, and it reconstructs the samples using the codebooks. In the section devoted to the experimental evaluation, we will discuss the performance of FAISS in comparison with our approach.

Table 1: Notation used throughout this manuscript

Symbol	**Definition**
$\mathcal{X}$	data domain
o, $o_i \in \mathcal{X}$	data objects
$\boldsymbol{v}$, $\boldsymbol{v}_i \in \mathbb{R}^D$	D-dimensional data vectors
$\boldsymbol{e} \in \mathbb{R}^D$	constant vector $[1, \ldots, 1]$
$q \in \mathcal{X}$, $\boldsymbol{q} \in \mathbb{R}^D$	query
$d(\cdot, \cdot)$	distance function $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$
$\ell_2(\cdot, \cdot)$, $\| \cdot \|_2$	Euclidean distance, Euclidean norm
$S_\rho(\cdot, \cdot)$	Spearman rho distance
$sim_{cos}(\cdot, \cdot)$	cosine similarity

(continued on next page)

Symbol	Definition
$\{\tau_1, \ldots, \tau_n\}$	codebook
$n \in \mathbb{N}$	codebook size
$f(\cdot)$	mapping of data objects into integer vectors (term frequencies)
$STR_f(o)$, t_o	surrogate text representations of the object o
$\cup$	space-separated concatenation operator
$\{p_1 \ldots p_n\} \subset \mathcal{X}$	set of pivots
$\Pi_o \in \mathbb{N}^n$	pivot permutation of $o \in \mathcal{X}$
$\Pi_o^{-1} \in \mathbb{N}^n$	inverted permutation of $o \in \mathcal{X}$
$\tilde{\Pi}_{\boldsymbol{v}}$	Deep Permutation of $\boldsymbol{v} \in \mathbb{R}^D$
$\tilde{\Pi}_{\boldsymbol{v}}^{-1} \in \mathbb{N}^n$	inverted Deep Permutation of $\boldsymbol{v} \in \mathbb{R}^D$
$k \in \mathbb{N}$	permutation-prefix length
$\tilde{\Pi}_{\boldsymbol{v},k}^{-1} \in \mathbb{N}^n$	inverted truncated Deep Permutation of $\boldsymbol{v} \in \mathbb{R}^D$
$R \in \mathbb{R}^{D \times D}$, $\boldsymbol{\mu} \in \mathbb{R}^D$	random orthogonal matrix, translation vector
$s \in \mathbb{R}$	scalar quantization factor ($s > 1$)
$\gamma \in \mathbb{N}$	thresholding parameter
$\boldsymbol{v}^+ \in \mathbb{R}^{2D}$	CReLU of the vector $\boldsymbol{v}$
$M \in \mathbb{N}$	number of sub-vectors in which a vector is divided in the PQ approach
$N \in \mathbb{N}$	number of Voronoi cells used in the PQ approach
$P \in \mathbb{N}$	number of Voronoi cells used to probe the posting list in the PQ approach

(continued on next page)

Symbol	Definition
C	code size (memory occupied by a sample) in the PQ approach
S_{DB}	query selectivity (average fraction of database accessed per query)
C_{DB}	query cost (average number of bytes accessed per query)
B_{DB}	total size of the database in bytes
$\delta_i, \hat{\delta}_i \in \mathbb{R}$	density of i-th dimension of a set of data vectors
$n_i \in \mathbb{N}$	number of elements in the i-th posting list

3. Background

3.1. Deep Features

Recently, a new class of image descriptor, built upon Convolutional Neural Networks, have been used as an effective alternative to descriptors built using local features such as SIFT, ORB, BRIEF, etc. CNNs have attracted an enormous interest within the Computer Vision community because of the state-of-the-art results achieved in challenging image classification tasks such as the ImageNet Large Scale Visual Recognition Challenge (`http://www.image-net.org`). In computer vision, CNNs have been used to perform several tasks, including image classification, as well as image retrieval [25, 26] and object detection [27], to cite some. Moreover, it has been proved that the representations learned by CNNs on specific tasks (typically supervised) can be transferred successfully across tasks [25, 28]. The activation of neurons of specific layers, in particular the last ones, can be used as features to semantically describe the visual content of an image.

Tolias et al. [29] proposed the R-MAC (Regional Maximum Activations of Convolutions) feature representation, which encodes and aggregates several re-

gions of the image in a dense and compact global image representation. To compute an R-MAC feature, an input image is fed to a fully convolutional network pre-trained on ImageNet. The output of the last convolutional layer is max-pooled over different spatial regions at different position and scales, obtaining a feature vector for each region. These vectors are then ℓ_2-normalized, PCA-whitened, ℓ_2-normalized again, and finally aggregated by summing them together and ℓ_2-normalizing the final result. The obtained representation is an effective aggregation of local convolutional features at multiple position and scales that it can be compared with the cosine similarity function.

We extracted the R-MAC features using fixed regions at two different scales as proposed in [29] instead of using the region proposal network. Defined S as the size in pixel of the minimum side of an image, we computed the multi-resolution descriptor aggregating the ones extracted at $S = 550$, 800 and $1,050$, resulting in a dense 2048-dimensional real-valued image descriptor.

4. Surrogate Text Representation

As we explained in the introduction, we aim to index and search a data set of feature vectors by exploiting off-the-shelf text search engines. So our main goal is to define a family of transformations that map a feature vector into a textual representation without the need for tedious training procedures. Of course, we also require that such transformations preserve as much as possible the proximity relations between the data, i.e. similar feature vectors are mapped to similar textual documents.

This basic idea was firstly exploited in [11], where the authors defined the Surrogate Text Representation (STR) to represent a generic metric object, i.e. an object living in a space where a distance function is defined [30]. The STR of an object is a space-separated concatenation of some alphanumeric codeword selected from a pre-defined dictionary. The original approach uses a permutation-based indexing technique (further described in Section 4.1) to generate the textual encoding. Here we observe that a surrogate text representation for an object

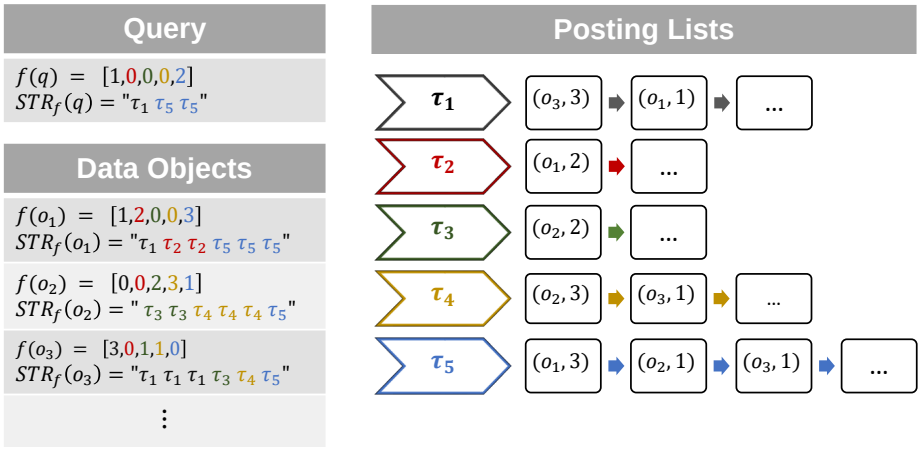


Figure 1: Example of the STR encodings for a query object q and three data objects o_1, o_2, o_3. The STRs are indexed using posting lists. The posting lists accessed at query time are the only ones associated to the codewords comparing in the surrogate text representation of the query. For example, given the query $STR_f(q)$ ="$\tau_1\ \tau_5\ \tau_5$", the posting lists accessed are just those related to the codewords τ_1 and τ_5.

o of a data domain $\mathcal{X}$ can be obtained more generally by defining a transformation Specifically, given a dictionary $\{\tau_1, \ldots, \tau_n\}$ and the transformation f, we define the surrogate text $t_o = STR_f(o)$ as

$$STR_f(o) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{f_i^{(o)}} \tau_i \tag{1}$$

where, by abuse of notation, we denote the space-separated concatenation of the codewords with the union operator $\cup$. Thus, by construction, the integer values of the i-th component of the vector $f(o)$, is the frequency of the codeword τ_i in the text $STR_f(o)$. For example, given $f(o) = [1, 3, 0, 2]$ and a codebook $\{\tau_1 =$ "A", $\tau_2 =$ "B", $\tau_3 =$ "C", $\tau_4 =$ "D"$\}$ we have $STR_f(o) =$ "A B B B D D".

The rationale of this approach is that a full-text search engine based on the *vector space model* [31] will generate a vector representation of the $STR_f(o)$ by counting the number of occurrences of the words in it, i.e. the term frequencies. In facts, these systems transform the text into vector representations using the well-known TF scheme and practically use the dot product as a function of

similarity between vectors. So, the abstract transformation $f(\cdot)$ represents a function that exactly generates the vectors that are internally represented by the search engine in the case of the simple term-weighting scheme. Using this representation, the search engine indexes the text by using inverted files, i.e. each object o is stored in the posting lists associated to the codewords appearing in $STR_f(o)$.

Ideally, the transformation $f(\cdot)$ should be

1. *order-preserving*: the ranked results to the query q (obtained by searching the original space $\mathcal{X}$) exactly correspond to the ranked results to the text query $STR_f(q)$ (obtained by searching the surrogate text space);[2]
2. *sparsifying*: for any $o \in \mathcal{X}$, the vector $f(o)$ is sparse.

The order-preserving property guaranties we do not loose any solutions when using the textual representation with respect to the results that would be obtained using the original data representation. However, having this property is quite impossible in practice since it is likely that any hand-crafted function $f(\cdot)$ introduces some approximations when transforming the data objects into term-frequency vectors. Thus, the *effectiveness* of a STR-based retrieval system highly depends on how good the transformation $f(\cdot)$ is in preserving the object similarities. The sparsifying property, instead, is requested for *efficiency* issues since each data object o will be stored in as many posting lists as the number of the non-zero elements in $f(o)$. Having dense vectors means that the search based on the inverted index would be significantly more expensive than the sequential scan, moreover also the index size would be much bigger than the original dataset. The ideal case is having uniformly sparse vectors so that the length of each posting list is approximately the same.

[2]Note that to compare a query with a set of objects we can use either a *similarity* or a *distance* function. In the former case, we search for the objects with the greatest similarity to the query. In the latter case, we search for the objects with the lowest distance from the query. A similarity function is said to be *equivalent* to a distance function if the ranked list of the results to a query is the same.

Unfortunately, the sparsifying property is in contrast with the order-preserving one since the former naturally introduces some approximations in the vectors, which basically translates our problem into a trade-off between effectiveness (score approximation) and efficiency (posting-list length).

In this paper, we are interested in indexing and searching *D-dimensional vectors* originally compared with the dot product. In this case, $\mathcal{X} = \mathbb{R}^D$ and the order-preserving property with respect to dot product can be expressed as

$$\forall\, \boldsymbol{q}, \boldsymbol{v_1}, \boldsymbol{v_2} \in \mathbb{R}^D \quad \boldsymbol{q} \cdot \boldsymbol{v_1} \leq \boldsymbol{q} \cdot \boldsymbol{v_2} \Rightarrow f(\boldsymbol{q}) \cdot f(\boldsymbol{v_1}) \leq f(\boldsymbol{q}) \cdot f(\boldsymbol{v_2}). \tag{2}$$

Below we present two text transformation techniques that can be used in this context: a Permutation-Based approach and Scalar Quantization approach. We then discuss how to achieve sparse representations for these two approaches.

4.1. Permutation-Based Approach

The basic idea of permutation-based indexing techniques is to represent feature objects as permutations of a set of identifiers. Traditionally, the permutation representations are built using the identifiers of a set of reference objects as permutants.

Formally, given a domain $\mathcal{X}$, a distance $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$, and a fixed set $\{p_1 \dots p_n\} \subset \mathcal{X}$ of *reference objects* (or *pivots*), we define a permutation-based representation Π_o (briefly *permutation*) of an object $o \in \mathcal{D}$ as the sequence of pivots identifiers sorted in ascending order by their distance from o [10, 32]. In other words, the permutation-based representation $\Pi_o = [\Pi_o(1), \dots, \Pi_o(n)]$ lists the pivot identifiers $\{1, 2, \dots, n\}$ in an order such that $\forall\, i \in \{1, \dots, n-1\}$,

$$d(o, p_{\Pi_o(i)}) < d(o, p_{\Pi_o(i+1)}) \tag{3}$$

or

$$\left[d(o, p_{\Pi_o(i)}) = d(o, p_{\Pi_o(i+1)})\right] \wedge \left[\Pi_o(i) < \Pi_o(i+1)\right]. \tag{4}$$

where $p_{\Pi_o(i)}$ indicates the pivot at position i in the permutation associated with the object o. An equivalent permutation-based representation is the *inverted*

permutation, defined as $\Pi_o^{-1} = [\Pi_o^{-1}(1), \Pi_o^{-1}(2), \ldots, \Pi_o^{-1}(n)]$, where $\Pi_o^{-1}(i)$ denotes the position of a pivot p_i in the permutation Π_o, so that $\Pi_o(\Pi_o^{-1}(i)) = i$. Thus, the coordinate i in the permutation Π_o is the identifier of the pivot at i-th position in the ranked list of the nearest pivots to o; the value at the coordinate i in the inverted representation Π_o^{-1} is the rank of the pivot p_i in the list of the nearest pivots to o.

The inverted representation of the permutations is often used in practice since it allows us to easily define most of the distance functions usually used to compare permutations, such as the Spearman rho and the Spearman Footrule distances [33].

The original STR approach [11] is based on a double transformation process. First, each data object o is transformed in a permutation vector (the inverted permutation) Π_o^{-1} using the approach presented above, and then it transforms the permutation into a textual representation t_o. The textual representation is obtained by associating each pivot p_i with an unique alphanumeric codeword τ_i and the permutation Π_o^{-1} with a sequence of codewords. Specifically, the text t_o is built in such a way that the occurrence of the codeword τ_i in it reflects the closeness of the pivot p_i to the object o: the lower the value $\Pi_o^{-1}(i)$ the higher the frequency of the term τ_i in the document t_o. Formally, the original STR approach uses the transformation $f_{perm} : o \mapsto n\boldsymbol{e} - \Pi_o^{-1}$, where $\boldsymbol{e} = [1, \ldots, 1]$ is the constant vector. Therefore,

$$STR_{f_{perms}}(o) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{n-\Pi_o^{-1}(i)} \tau_i. \tag{5}$$

The rationale behind this approach is that if two objects are very close one to the other, they will sort the set of pivots in a very similar way, and thus the corresponding permutations/text representations will be close as well. Note that we have no theoretical guarantees that the transformation

$$\Pi^{-1} : (\mathcal{X}, d) \to (\mathbb{N}^n, \ell_2) \tag{6}$$

$$o \mapsto \Pi_o^{-1} \tag{7}$$

is order-preserving in a strict sense, however several works [10, 34, 35] experimentally proved that the rankings obtained in the permutation space are good approximations of the rankings obtained in the original space. Moreover, it can be easily proved that the cosine similarity between any two term frequency vectors is a monotonic transformation of the squared Euclidean distance between the associated inverted permutations: $sim_{cos}(f_{perms}(o_1), f_{perms}(o_2)) = \alpha - \beta\, \ell_2(\Pi_{o_1}^{-1}, \Pi_{o_2}^{-1})^2$, where $\alpha \in R$, $\beta \in \mathbb{R}^+$ are constants (see [36, 37]). Therefore, a ranking obtained using the cosine similarity on term frequency vectors is equivalent to that obtained in the permutation space using the Spearman rho distance, which in turn is an approximation of the ranking obtained in the original data space.

So far, we have presented the general approach of STR based on the traditional permutation representations. However, when objects to be indexed are *real-valued vectors*, as in the case of deep features, we can exploit the *Deep Permutations* technique [12] that allows to generate the permutations at very low computational cost by avoiding the calculations of the distances between the pivots and the objects to be represented. Moreover, Amato et al.[12] shown that this encoding is more effective than the traditional permutation-based representation for both multimedia retrieval and similarity search tasks. In the Deep Permutations approach, the permutants are the indexes of the elements of the deep feature vectors rather that a predefined set of pivots. Specifically, the deep permutation of a feature vector $\boldsymbol{v} \in \mathbb{R}^n$ is obtained by sorting the *indexes* of the elements of $\boldsymbol{v}$, in descending order with respect to the corresponding element values. In other words, the Deep Permutation $\tilde{\Pi}_{\boldsymbol{v}} = [\tilde{\Pi}_{\boldsymbol{v}}(1), \ldots, \tilde{\Pi}_{\boldsymbol{v}}(n)]$ of a deep feature $\boldsymbol{v}$ is the permutation of the indexes $\{1, \ldots, n\}$ such that

$$\forall i = 1, \cdots, n-1, \quad \boldsymbol{v}(\tilde{\Pi}_{\boldsymbol{v}}(i)) \geq \boldsymbol{v}(\tilde{\Pi}_{\boldsymbol{v}}(i+1)), \tag{8}$$

where we use the notation $\boldsymbol{v}(j)$ to indicates the j-th element of $\boldsymbol{v}$. So the index i appears before index j in the permutation $\tilde{\Pi}_{\boldsymbol{v}}$ if the value $\boldsymbol{v}(i)$ is greater than or equal to $\boldsymbol{v}(j)$. Equivalently, using the inverted representation $\tilde{\Pi}_{\boldsymbol{v}}^{-1} = [\tilde{\Pi}_{\boldsymbol{v}}^{-1}(1), \ldots, \tilde{\Pi}_{\boldsymbol{v}}^{-1}(n)]$, we have that $\tilde{\Pi}_{\boldsymbol{v}}^{-1}(i) \leq \tilde{\Pi}_{\boldsymbol{v}}^{-1}(j)$ if $\boldsymbol{v}(i) \geq \boldsymbol{v}(j)$. Using

the approach introduced above, similarly to the traditional SRT technique, we can define the transformation $f_{DeepPerm} : \boldsymbol{v} \mapsto n\boldsymbol{e} - \tilde{\Pi}_{\boldsymbol{v}}^{-1}$ to associate a term frequency vector to each data object.

Suppose, for instance, that the feature vector is $\boldsymbol{v} = [0.1, 0.3, 0.4, 0, 0.2]$ (in reality, the number n of dimensions may be thousands). The deep permutation encoding of $\boldsymbol{v}$ is $\tilde{\Pi}_{\boldsymbol{v}} = [3, 2, 5, 1, 4]$, that is permutant (index) 3 is in position 1, permutant 2 is in position 2, permutant 5 is in position 3, etc. In facts, $\boldsymbol{v}(3) = 0.4$ is the biggest value in $\boldsymbol{v}$, $\boldsymbol{v}(2) = 0.3$ is the second biggest element value, and so on. The corresponding inverted deep permutation is $\tilde{\Pi}_{\boldsymbol{v}}^{-1} = [4, 2, 1, 5, 3]$, that is permutant (index) 1 is in position 4, permutant 2 is in position 2, permutant 3 is in position 1, etc. The term frequencies vector will be $f_{DeepPerm}(\boldsymbol{v}) = [1, 3, 4, 0, 2]$, and thus $STR_{f_{DeepPerms}}(\boldsymbol{v}) =$ "$A\,B\,B\,B\,C\,C\,C\,C\,E\,E$".

4.2. Scalar Quantization-Based Approach

In this section, we propose an alternative approach to provide a surrogate text representation for feature vectors. The idea behind our Scalar Quantization approach is to map the real-valued vector components independently into a smaller set of integer values which act as the term frequencies of a predefined set of codewords. However, as for more generic Product Quantization, the scalar quantization has mainly the purpose of making a compact representation suitable for approximate nearest neighbor search.

The first step in our approach is the application of an order-preserving transformation to the vectors To understand why, remember that in our approach each component of the vectors is associated with a posting list in which each post stores the *id* of the vector and the value of the component itself, if nonzero. Therefore, if on average some component is nonzero for many data vectors then the corresponding posting list will be accessed many times, provided that the queries follow the same distribution of the data. The ideal case occurs when the component share exactly the same distribution (same mean and variance is sufficient). In this way, we try to increase to cases where the dimensional components of the features vectors have same mean and variance, with mean

equal to zero. This last constraint is because we want then sparsify the vectors, but this aspect is further investigated in the next subsection.

$$\forall\, \boldsymbol{q}, \boldsymbol{v_1}, \boldsymbol{v_2} \in \mathbb{R}^D \quad \boldsymbol{q} \cdot \boldsymbol{v_1} \leq \boldsymbol{q} \cdot \boldsymbol{v_2} \Rightarrow R\boldsymbol{q} \cdot R(\boldsymbol{v_1} - \boldsymbol{\mu}) \leq R\boldsymbol{q} \cdot R(\boldsymbol{v_2} - \boldsymbol{\mu}). \quad (9)$$

. The next step is transforming the rotated vectors into term frequency vectors. We do it by quantizing the vectors so that posting entries will contain numeric values proportional to the float values of the deep feature entries. Specifically, we use the transformation $\boldsymbol{w} \rightarrow \lfloor s\boldsymbol{w} \rfloor$ where $\lfloor \cdot \rfloor$ denotes the floor function and s is a multiplication factor > 1 that works as a *quantization factor*. This process introduces a quantization error due to the representation of float components in integers. However, as we will see, this error does not affect the retrieval effectiveness.

In summary, the Scalar Quantization-based STR is obtained using the transformation $f_{ScalarQuant} : \boldsymbol{v} \mapsto \lfloor sR(\boldsymbol{v} - \boldsymbol{\mu}) \rfloor$.

For instance, suppose after the random orthogonal transformation we have the feature vector $\boldsymbol{v} = [0.1, 0.3, 0.4, 0, 0.2]$, by adopting a multiplication factor $s = 10$, we obtain the term frequencies vector will be $f_{ScalarQuant}(\boldsymbol{v}) = [1, 3, 4, 0, 2]$, and thus $STR_{f_{ScalarQuant}}(\boldsymbol{v}) =$ "$A\,B\,B\,B\,C\,C\,C\,C\,E\,E$".

4.3. Sparsification

The two approaches presented above are intended to encode a vector of real numbers into a vector of integers preserving as much as possible the order with respect to the dot product. However, this approach does not solve the problem that in most cases these vectors are dense, which means having low efficiency performance if the inverted files are used to index the textual documents. In order to sparsify the term frequency vectors, that is to cancel the less significant components of the vectors, we must accept a further loss in precision. To achieve this we propose two ways: keep the top-k larger components of the vector or maintain components above a certain threshold $1/\gamma$ by zeroing all the others. Where $\gamma \in \mathbb{N}$ is the parameter we use to control the sparseness of the thresholded

feature. The first approach (*top-k*) is better suited to the permutation-based approach and the latter (*thresholding*) to the scalar quantization approach.

The top-k approach on the permutation-based encoding can be seen as the selection of a fixed-length prefix of the permutations. This approach works well since it is assumed that the greatest information is in the first k elements of the permutation, i.e. the identifiers of the closest pivots to the object to be represented. It is equivalent to say that instead of using the full-length inverted permutation $\tilde{\Pi}_{\boldsymbol{v}}^{-1}$ we use the *truncated inverted permutation*:

$$\tilde{\Pi}_{\boldsymbol{v},k}^{-1}(i) = \begin{cases} \tilde{\Pi}_{\boldsymbol{v}}^{-1}(i) & \text{if} \quad \tilde{\Pi}_{\boldsymbol{v}}^{-1}(i) \leq k \\ k+1 & \text{otherwise} \end{cases}. \tag{10}$$

Note that, in the permutation-based indexing context, using the positions of the nearest k out of n pivots often leads to obtaining better or similar effectiveness then using the full-length permutation [10, 32], resulting also in a more compact data encoding. To transform the truncated inverted permutation into a term frequency vectors we use $f_{DeepPerm,\,k} : \boldsymbol{v} \mapsto (k+1)\boldsymbol{e} - \tilde{\Pi}_{\boldsymbol{v},k}^{-1}$.

The thresholding adopted for scalar quantization simply discards by setting to zero the dimension of the vector $\boldsymbol{v}$ having absolute values above a specified threshold:

$$\boldsymbol{v}_\gamma(i) = \begin{cases} \boldsymbol{v}(i) & \text{if} \quad \boldsymbol{v}(i) \geq \frac{1}{\gamma} \\ 0 & \text{otherwise} \end{cases}, \tag{11}$$

where $\boldsymbol{v}(i)$ indicates the i-th dimension of the feature vector $\boldsymbol{v}$. This approach is optimal when we have many components near or equal to zero;

4.4. Dealing with Negative Values

In order to index R-MAC features and represent them in the vector space model of text retrieval, we encode each dimension of the R-MAC features as a different codeword, and we use the TF field to represent a single value of our feature vector. However, TF must be positive (no search engine admits negative TFs even if this in principle would be possible), nonetheless, both negative and positive elements contribute to informativeness. In the two approaches

presented above, this produces two different negative effect, but fortunately the solution we adopt is the same for both. In the scalar quantization case, negative quantized components are not admitted by construction; in the deep permutation-based approach, negative components are neglected since they always fall behind the positive ones in the ordering. Naive techniques such as truncation to zero or taking the absolute value result in a degraded performance due to respectively loss or aliasing of information. In order to prevent this imbalance towards positive activations at the expense of negative ones, we use the Concatenated Rectified Linear Unit (CReLU) transformation [38]. It simply makes an identical copy of vector elements, negate it, concatenate both original vector and its negation, and then apply ReLU altogether. More precisely the CReLU of the vector $\boldsymbol{v}$ is defined as $\boldsymbol{v}^+ = ReLU([\boldsymbol{v}, -\boldsymbol{v}])$, where the $ReLU(\cdot) = max(\cdot, 0)$ is applied element-wise. After applying CReLU, we applied either $f_{DeepPerm}$ or $f_{ScalarQuant}$ (or their sparsifying versions) to $\boldsymbol{v}^+$ as described in the previous sections.

For instance, if $\boldsymbol{v} = [0.1, -0.3, -0.4, 0, 0.2]$, the CReLU applied on $\boldsymbol{v}$ is $\boldsymbol{v}^+ = [0.1, 0, 0, 0, 0.2, 0, 0.3, 0.4, 0, 0]$. If using the Deep Permutations approach, the corresponding inverted permutation is $\Pi^{-1}_{\boldsymbol{v}^+} = [4, 5, 6, 7, 3, 8, 2, 1, 9, 10]$ (ties take random positions) and thus $f_{DeepPerm}(\boldsymbol{v}^+) = [6, 5, 4, 3, 7, 2, 8, 9, 1, 0]$. For completeness we report the example also for the case of a spasifying factor $k = 4$, for which we have $\Pi^{-1}_{\boldsymbol{v}^+,4} = [4, 5, 5, 5, 3, 5, 2, 1, 5, 5]$ and $f_{DeepPerm,\,k}(\boldsymbol{v}^+) = [1, 0, 0, 0, 2, 0, 3, 4, 0, 0]$. If we apply the Scalar Quantization technique to the same vector $\boldsymbol{v}^+$, with the values of $\gamma = 5$ and $s = 10$, we obtain $f_{DeepPerm,\,k}(\boldsymbol{v}^+) = [0, 0, 0, 0, 2, 0, 3, 4, 0, 0]$.

Notice that in general, the CReLU operation is lossy since the dot product between vectors is not preserved, i.e., $\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 \leq \boldsymbol{v}_1^+ \cdot \boldsymbol{v}_2^+$. However, this transformation allows us to apply the deep permutation approach without completely neglecting the negative activations of the R-MAC features. Moreover, the duplication of the dimensionality introduced by the CReLU does not increase the number of non-zero elements (half the elements are always zero) and therefore neither the space occupied in the inverted index.

5. Experimental Evaluation

The aim of this section is to assess the performance of the proposed solution in a content-based retrieval task both in terms of effectiveness and efficiency. To this end, we are able to evaluate the approximation introduced with respect to the exact similarity search algorithm against the impact of this approximation with respect to the user perception of the retrieval task. We extracted the R-MAC features from the images of two different benchmarks: *INRIA Holidays* and *Oxford Buildings*. INRIA Holidays [39] is a standard benchmark for image retrieval consisting in a collection of 1,491 images representing a large variety of scene type (natural, man-made, water, etc). The authors of INRIA Holidays selected 500 queries and manually identified a list of qualified results for each of them. In the literature, this benchmark is extended with the distractor dataset MIRFlickr including 1M images called MIRFlickr1M (`http://press.liacs.nl/mirflickr/`). Oxford Buildings [20] is composed of 5,062 images of 11 Oxford landmarks downloaded from Flickr. A manually labeled groundtruth is available for five queries for each landmark, for a total of 55 queries. As for *INRIA Holidays*, we merged the dataset with the distraction dataset Flickr100k including 100k images [3].

In the implementation of FAISS, the number of Voronoi cells N controls the number and length of the postings lists, while the number of PQ sub-quantizer M controls the amount of memory occupied by a sample, also known as code size C. Moreover, we directly choose the code size C, and M is set accordingly by the software. Since FAISS is an in-memory index, C is usually set to the maximum code size to fit the whole dataset we want to index in main memory. While N and C are fixed parameter chosen at indexing time, the number of nearest neighbor Voronoi cells to probe P can be adjusted at query time and can be tuned to control the effectiveness-efficiency trade-off.

For a fair comparison, we used a configuration for FAISS which gives the

[3] `http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/`

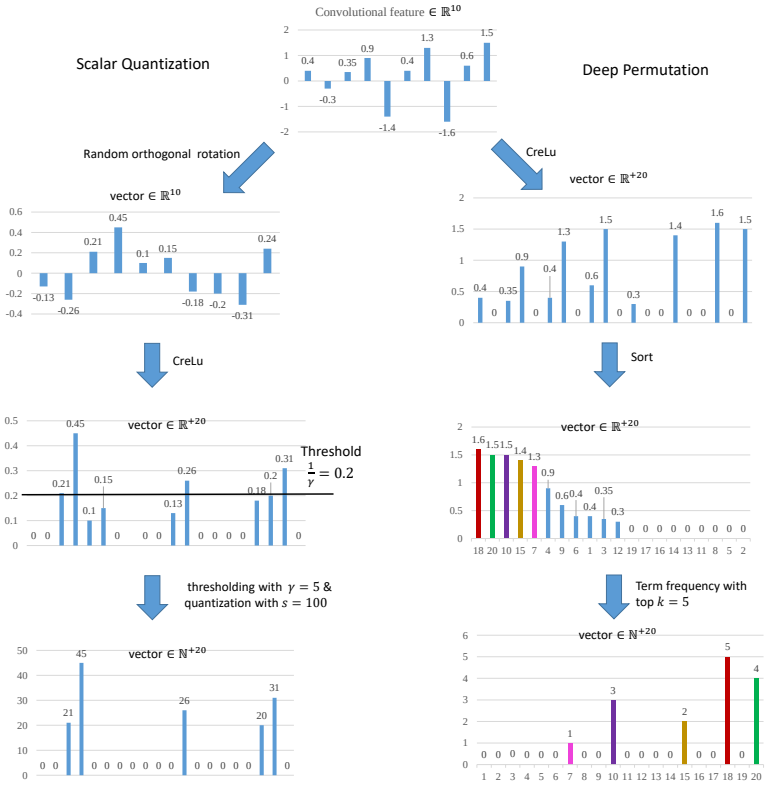


Figure 2: Comparison between the of generation Deep Permutaion and Scalar Quantization from the same feature vector.

best trade-off between effectiveness and efficiency for each dataset, choosing a relatively big code size ($C = 1,024$ for Holidays, $C = 512$ for Oxford) and optimal number of Voronoi cells ($N = 16$k for Holidays and $N = 1.5$k for Oxford). However, these configurations have long index training times and a large memory footprint (for 1M images and $C = 1,024$, FAISS requires about 1GB in main memory, against about 0.7GB of our solution in secondary memory in the larger configuration for $k = 400$), and this could limit its scalability, especially in systems with limited main memory.

We generated different sets of permutations from the original features using different values of k for the permutation-based approach (i.e., we consider top-k truncated permutations) and γ for the scalar quantization approach. For each configuration, we measured the mAP (mean Average Precision) obtained and the query execution times for each configuration. As the goal of our approach is to work in a context of large scale image search, in the experiments, we report the mAP (on the y-axis) in function of either the *Query Selectivity* S_{DB}, i.e. the average fraction of database accessed per query, or the *Query Cost* C_{DB}, i.e. average number of bytes accessed per query. If the total size of the database in bytes is B_{DB}, then the following equality holds: $C_{DB} = B_{DB}\, S_{DB}$.

The query selectivity for surrogate text representation techniques can easily be derived as follows. Given a database comprised of N D-dimensional objects ($D = 2,048$ for $\boldsymbol{v}$ in the case of R-MAC), the query selectivity S_{DB} is defined as the fraction of the $N \cdot D$ elements accessed to answer a query. Let δ_i be the fraction of samples in the database having a non-zero element in its i-th dimension. Given a query $\boldsymbol{q}$, we access the i-th posting list only if the i-th dimension of $\boldsymbol{q}$ is non-zero, which is true with probability δ_i since query and dataset objects share the same distribution. The number n_i of elements contained in the i-th posting list equals the number of elements of the database having non-zero i-th dimension, that is $n_i = \delta_i\, N$. Hence, the number of elements accessed to answer a query is

$$\sum_{i=1}^{D} \delta_i\, n_i\,, \tag{12}$$

because i-th posting list contains n_i elements, and we access it with δ_i probability. Thus, we can compute the query selectivity

$$S_{DB} = \frac{1}{ND} \sum_{i=1}^{D} \delta_i \, n_i = \frac{1}{D} \sum_{i=1}^{D} {\delta_i}^2 \,, \tag{13}$$

Similarly, in the case of the CReLU-ed vector $\boldsymbol{v}^+$, the number of dimensions (and thus posting lists) doubles, but the number of total non-zero elements stored in the index is still at most $N \cdot D$. Thus,

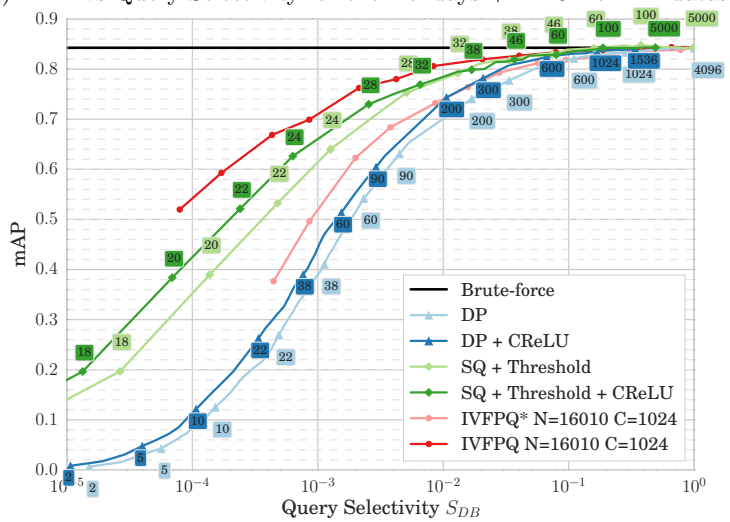
$$S_{DB} = \frac{1}{D} \sum_{i=1}^{2D} {\hat{\delta}_i}^2 \,. \tag{14}$$

Note that, in the surrogate text representation, we associate a distinct codeword to each dimension of the vector, and each codeword is associated to a posting list. An object o is stored in the posting list corresponding to the codeword τ_i if and only if the occurrence of τ_i in the surrogate text representation of the object o is greater than zero. At query time, we access only to the posting lists associated to the codewords appearing in the surrogate text representation of the query object, and thus, the more balanced are the posting lists of the inverted index, the smaller the query selectivity is. The greater is k, the lower is the sparsity, and hence the greater is the mAP and the query selectivity. Since we are dealing with an approximate approach, the brute-force mAP, i.e. the one computed with the original features and a sequential scan of the entire database, can be considered as an upper-bound.

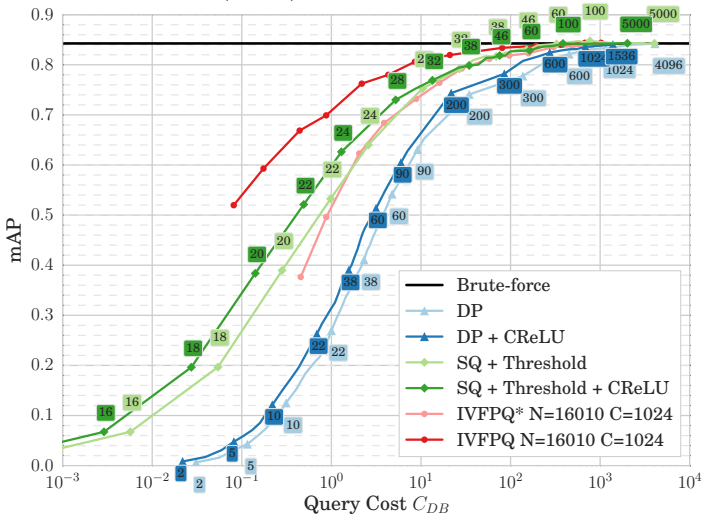
Figure 3 reports the performance of the evaluated approaches on both datasets. We can see that we reach satisfactory levels of effectiveness for a query selectivity of 10^{-2}. Moreover, we have shown the remarkable advantage of the CReLU preprocessing on R-MAC vectors in comparison with the deep permutation method applied directly on the original vectors.

We would like to stress two points concerning the experiments on scalar quantization. First, we used a large value of the scalar quantization factor s in order to exploit the full range of numbers that can be expressed in term-frequencies integers depending on the search engine implementation employed

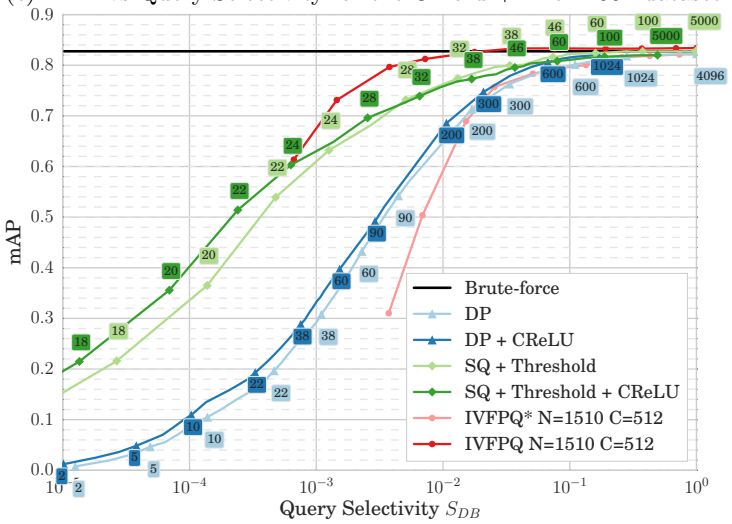
(a) mAP vs Query Selectivity for the Holidays + MIRFlickr1M dataset.



(b) mAP vs Query Cost (Bytes) for the Holidays + MIRFlickr1M dataset.



(c) mAP vs Query Selectivity for the Oxford + Flickr100k dataset.



(d) mAP vs Query Cost (Bytes) for the Oxford + Flickr100k dataset.

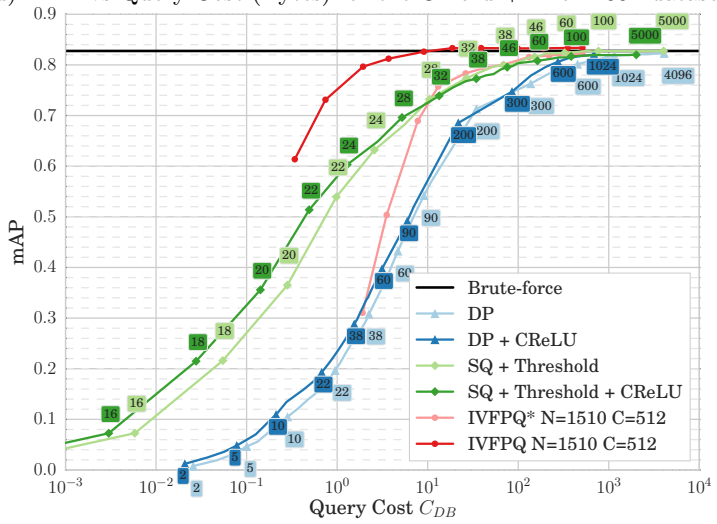


Figure 3: Comparison of the performance of our methods (SQ and DP) and PQ-compressed/inverted-file-based FAISS indexes (IVFPQ and IVFPQ*) on Holidays + MIR-Flickr1M (a-b) and on Oxford + Flickr100k (c-d), in terms of mAP and query selectivity/cost trade-off. Note that IVFPQ and IVFPQ* are obtained using two different dataset for the cood-ebook learning: the indexed dataset itself and the T4SA dataset, respectively. The curves are obtained varying k (for DP), γ (for thresholded scalar quantization), and the number of Voronoi cell accessed P (for IVFPQ/IVFPQ*). Values of k and γ are reported near each point. The horizontal line represents the mAP obtained using the original R-MAC vectors and performing a sequential scan of all the dataset (brute-force approach).

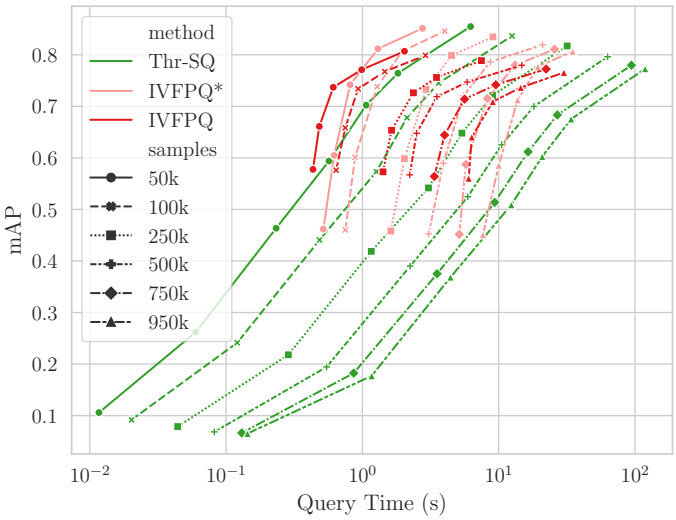
(we considered a 32-bit integer in our experiments on Elasticseach). We could analyze the impact on performance for low values of s, but we believe that this study is of little relevance because the SRT approach involves using a standard search engine, so we would not have any control over how the integers within the posting lists are effectively implemented by the search engine. Second, the reported experiments with Scalar Quantization without the CReLU transformation ('SQ + Threshold' line in Figure 3) are simulated, since the components of the vectors can also be negative and thus not implementable with the SRT approach on a standard textual search engine.

We also report the performance of FAISS indexes when varying the number of Voronoi cells probed P. We report two curves related to FAISS which correspond to two different datasets used for learning the codebooks: the indexed dataset itself, on which the mAP evaluation is performed, and an uncorrelated dataset of images. For the latter, we employed the T4SA dataset [40], a large collection of images collected from the live stream of random 5% of global tweets using the Twitter Streaming API. The reason for this choice is to show how the performance of FAISS is sensitive to the specific dataset distribution on which it has been trained. Indeed, we see the impact of this aspect is really strong, and it could, in real applications, influence the scalability of the system or require continuous codebook adjustments, forcing to re-indexing the data periodically. Our solution has an intermediate performance but does not require any training procedure and therefore any re-adjustments.

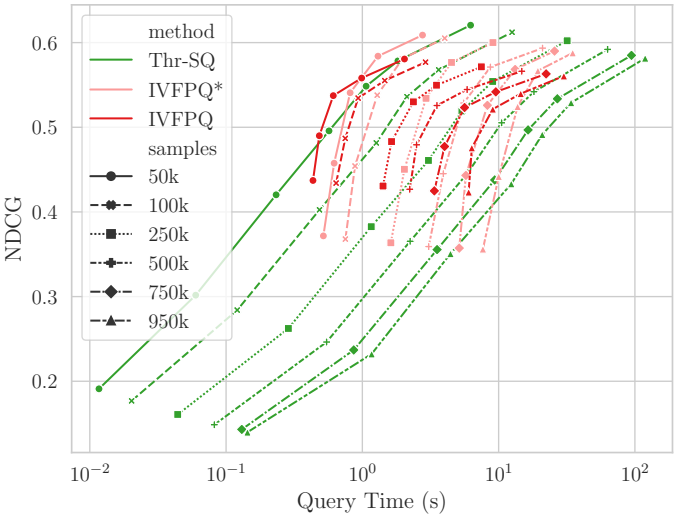
6. Conclusions and Future Works

This paper has proposed a simple and effective methodology to index and retrieve convolutional features without the need for a time-consuming codebook learning step. To get an idea, consider that FAISS takes about three hours for learning the codebook from about a million R-MAC features with the configuration used in the experiments. This learning step is often left out by the authors; nevertheless, in the context of Web-scale applications in which we

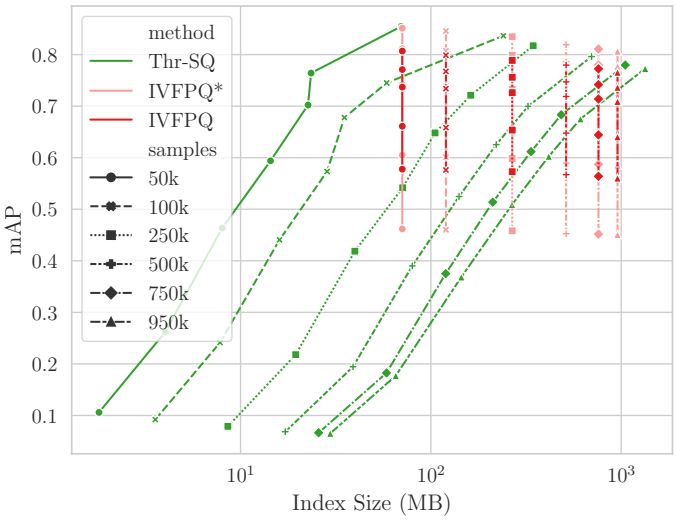
(a) mAP vs Query Time (s) for the Holidays + MIRFlickr1M dataset.



(b) NDCG vs Query Time (s) for the Holidays + MIRFlickr1M dataset.



(c) mAP vs Index Size (Bytes) for the Holidays + MIRFlickr1M dataset.



(d) NDCG vs Index Size (Bytes) for the Holidays + MIRFlickr1M dataset.

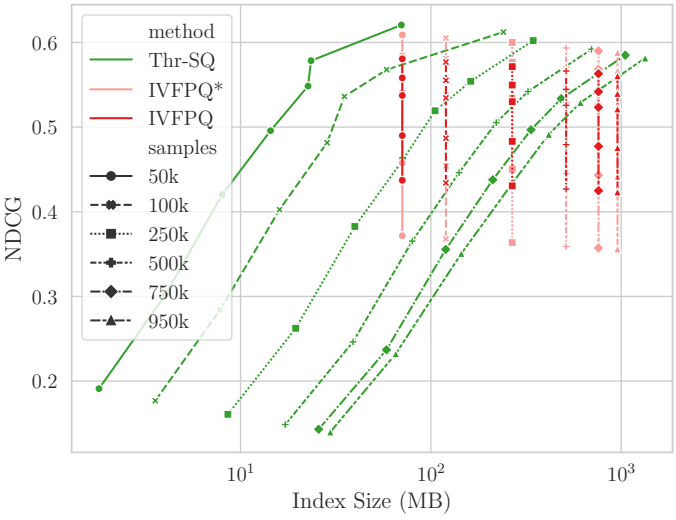


Figure 4: Effectiveness (mAP and NDCG) versus time (average query time, a-b) and space (index size, c-d) resources for our best method (Thresholded SQ with CReLU) and PQ-compressed/inverted-file-based FAISS indexes (IVFPQ and IVFPQ*) on Holidays + MIRFlickr1M. IVFPQ and IVFPQ* are obtained using two different dataset for the coodebook learning: the indexed dataset itself and the T4SA dataset, respectively. The curves are obtained varying γ (for thresholded scalar quantization), and the number of Voronoi cell accessed P (for IVFPQ/IVFPQ*).

frame our work, we think that not only processing time is a relevant aspect, but also being as independent as possible from data distribution during the indexing phase is a crucial point to be robust to data distribution drifting through time.

Another important aspect of our work is that our technique is completely independent from the technology used for indexing. We can rely on the latest text search engine technologies, without having to worry about issues related to implementation problems, such as software maintenance, updates to new hardware technologies, bugs, etc. Furthermore, with our approach, it is possible to include in the image records, in addition to the visual features (which are in textual form), other information such as text metadata, geotags, etc.

We plan to extend our work by investigating the possibility of modifying the R-MAC extraction network to directly learn a "discretized vector" similar to the one we get with the proposed hand-crafted method while trying to maximize sparsity and increase the cardinality of the codebook as much as possible.

7. Acknowledgements

The work was partially supported by Smart News, "Social sensing for breaking news", CUP CIPE D58C15000270008, by VISECH, ARCO-CNR, CUP B56J17001330004, and by Automatic Data and documents Analysis to enhance human-based processes (ADA), CUP CIPE D55F17000290009, and by the AI4EU project, funded by the EC (H2020 - Contract n. 825619). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research.

References

[1] J. Zobel, A. Moffat, Inverted files for text search engines, ACM computing surveys (CSUR) 38 (2) (2006) 6. doi:10.1145/1132956.1132959.

[2] D. Arroyuelo, M. Oyarzún, S. González, V. Sepulveda, Hybrid compression of inverted lists for reordered document collections, Information Processing

& Management 54 (6) (2018) 1308–1324. doi:10.1016/j.ipm.2018.05.007.

[3] F. Lashkari, E. Bagheri, A. A. Ghorbani, Neural embedding-based indices for semantic search, Information Processing & Management 56 (3) (2019) 733–755. doi:10.1016/j.ipm.2018.10.015.

[4] S. Pandey, P. Khanna, H. Yokota, A semantics and image retrieval system for hierarchical image databases, Information Processing & Management 52 (4) (2016) 571–591. doi:10.1016/j.ipm.2015.12.005.

[5] D. Novak, M. Batko, P. Zezula, Large-scale similarity data management with distributed metric index, Information Processing & Management 48 (5) (2012) 855–872. doi:10.1016/j.ipm.2010.12.004.

[6] R. Liu, S. Wei, Y. Zhao, Y. Yang, Indexing of the CNN features for the large scale image search, Multimedia Tools and Applications 77 (24) (2018) 32107–32131. doi:10.1007/s11042-018-6210-3.

[7] E. Mohedano, K. McGuinness, N. E. O'Connor, A. Salvador, F. Marques, X. Giro-i Nieto, Bags of local convolutional features for scalable instance search, in: Proceedings of the ACM International Conference on Multimedia Retrieval, ICMR 2016, ACM, 2016, pp. 327–331. doi:10.1145/2911996.2912061.

[8] J. Johnson, M. Douze, H. Jégou, Billion-scale similarity search with gpus, CoRR abs/1702.08734. arXiv:1702.08734.
URL http://arxiv.org/abs/1702.08734

[9] G. Navarro, N. Reyes, New dynamic metric indices for secondary memory, Information Systems 59 (2016) 48–78. doi:10.1016/j.is.2016.03.009.

[10] G. Amato, C. Gennaro, P. Savino, MI-File: using inverted files for scalable approximate similarity search, Multimedia Tools and Applications 71 (3) (2014) 1333–1362. doi:10.1007/s11042-012-1271-1.

[11] C. Gennaro, G. Amato, P. Bolettieri, P. Savino, An approach to content-based image retrieval based on the lucene search engine library, in: Proceedings of the International Conference on Theory and Practice of Digital Libraries, TPDL 2010, Springer Berlin Heidelberg, 2010, pp. 55–66. doi:10.1007/978-3-642-15464-5_8.

[12] G. Amato, F. Falchi, C. Gennaro, L. Vadicamo, Deep Permutations: Deep convolutional neural networks and permutation-based indexing, in: Proceedings of the 9th International Conference on Similarity Search and Applications, SISAP 2016, LNCS, Springer International Publishing, 2016, pp. 93–106. doi:10.1007/978-3-319-46759-7_7.

[13] G. Amato, F. Carrara, F. Falchi, C. Gennaro, Efficient indexing of regional maximum activations of convolutions using full-text search engines, in: Proceedings of the ACM International Conference on Multimedia Retrieval, ICMR 2017, ACM, 2017, pp. 420–423. doi:10.1145/3078971.3079035.

[14] G. Amato, P. Bolettieri, F. Carrara, F. Falchi, C. Gennaro, Large-scale image retrieval with elasticsearch, in: Proceeding of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, ACM, 2018, pp. 925–928. doi:10.1145/3209978.3210089.

[15] G. Amato, F. Debole, F. Falchi, C. Gennaro, F. Rabitti, Large scale indexing and searching deep convolutional neural network features, in: Proceedings of the International Conference on Big Data Analytics and Knowledge Discovery, DaWaK 2016, Springer International Publishing, 2016, pp. 213–224. doi:10.1007/978-3-319-43946-4_14.

[16] L. Zheng, Y. Yang, Q. Tian, SIFT meets CNN: A decade survey of instance retrieval, IEEE Transactions on Pattern Analysis and Machine Intelligence 40 (5) (2018) 1224–1244. doi:10.1109/TPAMI.2017.2709749.

[17] T. A. Nakamura, P. H. Calais, D. de Castro Reis, A. P. Lemos, An anatomy

for neural search engines, Information Sciences 480 (2019) 339–353. doi:10.1016/j.ins.2018.12.041.

[18] A. Arampatzis, K. Zagoris, S. A. Chatzichristofis, Dynamic two-stage image retrieval from large multimedia databases, Information Processing & Management 49 (1) (2013) 274–285. doi:10.1016/j.ipm.2012.03.005.

[19] J. Sivic, A. Zisserman, Video Google: A text retrieval approach to object matching in videos, in: Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2, ICCV 2003, IEEE, 2003, pp. 1470–1477. doi:10.1109/ICCV.2003.1238663.

[20] J. Philbin, O. Chum, M. Isard, J. Sivic, A. Zisserman, Object retrieval with large vocabularies and fast spatial matching, in: Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2007, IEEE, 2007, pp. 1–8. doi:10.1109/CVPR.2007.383172.

[21] D. Novak, M. Batko, P. Zezula, Large-scale image retrieval using neural net descriptors, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2015, ACM, 2015, pp. 1039–1040. doi:10.1145/2766462.2767868.

[22] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, J. Sivic, NetVLAD: CNN architecture for weakly supervised place recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, IEEE, 2016, pp. 5297–5307. doi:10.1109/CVPR.2016.572.

[23] J. Yue-Hei Ng, F. Yang, L. S. Davis, Exploiting local features from deep networks for image retrieval, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPRW 2015, IEEE, 2015, pp. 53–61. doi:10.1109/CVPRW.2015.7301272.

[24] H. Jégou, M. Douze, C. Schmid, Product quantization for nearest neighbor search, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (1) (2011) 117–128. doi:10.1109/TPAMI.2010.57.

[25] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, T. Darrell, DeCAF: A deep convolutional activation feature for generic visual recognition, CoRR abs/1310.1531. arXiv:1310.1531.

[26] A. Babenko, A. Slesarev, A. Chigorin, V. Lempitsky, Neural codes for image retrieval, in: Proceedings of 13th European Conference on Computer Vision, ECCV 2014, Springer, 2014, pp. 584–599. doi:10.1007/978-3-319-10590-1_38.

[27] R. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, IEEE, 2014, pp. 580–587. doi:10.1109/CVPR.2014.81.

[28] A. S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson, CNN features off-the-shelf: an astounding baseline for recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPRW 2014, IEEE Computer Society, 2014, pp. 512–519. doi:10.1109/CVPRW.2014.131.

[29] G. Tolias, R. Sicre, H. Jégou, Particular object retrieval with integral max-pooling of CNN activations, CoRR abs/1511.05879. arXiv:1511.05879. URL http://arxiv.org/abs/1511.05879

[30] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search: The Metric Space Approach, Vol. 32 of Advances in Database Systems, Springer-Verlag, 2006.

[31] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Inc., New York, NY, USA, 1986.

[32] G. Amato, F. Falchi, F. Rabitti, L. Vadicamo, Some Theoretical and Experimental Observations on Permutation Spaces and Similarity Search, SISAP 2014, LNCS, Springer International Publishing, 2014, pp. 37–49. doi:10.1007/978-3-319-11988-5_4.

[33] P. Diaconis, Group representations in probability and statistics, Lecture notes-monograph series 11 (1988) i–192.

[34] E. Chavez, K. Figueroa, G. Navarro, Effective proximity retrieval by ordering permutations, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (9) (2008) 1647 –1658. doi:10.1109/TPAMI.2007.70815.

[35] A. Esuli, MiPai: Using the PP-Index to Build an Efficient and Scalable Similarity Search System, in: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, SISAP 2009, 2009, pp. 146–148. doi:10.1109/SISAP.2009.14.

[36] G. Amato, P. Bolettieri, F. Falchi, C. Gennaro, L. Vadicamo, Using Apache Lucene to Search Vector of Locally Aggregated Descriptors, in: Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications-Volume 4: VISAPP, VISIGRAPP 2016, SciTePress, 2016, pp. 383–392. doi:10.5220/0005722503830392.

[37] L. Vadicamo, Enhancing content-based image retrieval using aggregation of binary features, deep learning, and supermetric search, Ph.D. thesis, University of Pisa, PhD in Ingegneria dell'Informazione (5 2018).
URL https://etd.adm.unipi.it/theses/available/etd-04242018-161334/

[38] W. Shang, K. Sohn, D. Almeida, H. Lee, Understanding and improving convolutional neural networks via concatenated rectified linear units, in: Proceedings of the 33rd International Conference on Machine Learning, Vol. 48 of ICML 2016, JMLR.org, 2016, pp. 2217–2225.

[39] H. Jégou, M. Douze, C. Schmid, Hamming embedding and weak geometric consistency for large scale image search, in: Proceedings of the European Conference on Computer Vision, Vol. 5302 of ECCV 2008, LNCS, Springer, 2008, pp. 304–317. doi:10.1007/978-3-540-88682-2_24.

[40] L. Vadicamo, F. Carrara, A. Cimino, S. Cresci, F. Dell'Orletta, F. Falchi, M. Tesconi, Cross-media learning for image sentiment analysis in the wild, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, ICCVW 2017, 2017, pp. 308–317. doi:10.1109/ICCVW.2017.45.