The Yael library
Matthijs Douze, Hervé Jégou

To cite this version:
Matthijs Douze, Hervé Jégou. The Yael library. 22nd ACM International Conference on Multimedia, Nov 2014, Orlando, United States. pp.687-690, 10.1145/2647868.2654892 . hal-01020695

HAL Id: hal-01020695
https://inria.hal.science/hal-01020695
Submitted on 8 Jul 2014

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.
Abstract

This paper introduces YAEL, a library implementing computationally intensive functions used in large scale image retrieval, such as neighbor search, clustering and inverted files. The library offers interfaces for C, Python and Matlab. Along with a brief tutorial, we analyze and discuss some implementation choices and their impact on efficiency. http://yael.gforge.inria.fr/

1. Motivation and overview

The motivation of YAEL is twofold: We aim at providing

- core and optimized instructions and methods commonly used for large-scale multimedia retrieval systems;
- more sophisticated functions associated with state-of-the-art methods, such as the Fisher vector [11, 10], VLAD [8, 10], Hamming Embedding [5, 6] or more generally methods based on inverted file systems, such as selective match kernels [13].

Having these two kinds of functions implemented in the core library, we can rapidly disseminate open source packages reproducing the results of recent papers for image retrieval. YAEL is intended as an API and does not implement a retrieval system in an integrated manner: only a few test programs are available for key tasks such as k-means. Yet this can be done on top of it with a few dozen lines of Matlab or Python code.

We provide one sample package relying on YAEL, which implements an image retrieval system based on the Fisher vector. Yet we mention that several of such packages are available online, each being associated with a particular task and published method. Section 3 lists a few of these, along with download statistics.

Design choices. YAEL is designed to handle dense data in float, as it is primarily used for signal processing tasks where the quality of the representation is determined by the number of dimensions rather than the precision of the components. In the Matlab interface, single matrices, and float32 in Python.

YAEL was designed initially to manipulate matrices in C. It was interfaced for Python using SWIG, which gives low-level access to the full library. An additional Numpy layer (ynumpy) is provided for high-level functions. The most important functions of YAEL are wrapped in Mex to be callable from Matlab.

Performance is very important. YAEL has computed k-means with hundreds of thousand centroids and routinely manipulate matrices that occupy more than 1/2 the machine’s RAM. This means it is lightweight and 64-bit clean.

The design choices of YAEL are governed by efficiency concerns more than by portability. The library is maintained for Linux and MacOS. Although an older version of the library was ported to Windows OS, only a few functions of the Matlab interface work on this platform.

Dependencies. YAEL relies on as few external libraries as possible. The only mandatory ones are BLAS/Lapack (for performance) and the Python/C API. Other libraries (Matlab’s mex, Arpack, OpenMP) are optional.

Threading. Functions of significant complexity are parallelized. The level at which threading occurs must be chosen carefully: at the BLAS level, the YAEL function or the calling function. Therefore, YAEL functions often take a nt parameter, the number of threads.

Interface. YAEL contains hundreds of functions at the C level. Only a few are exported in Matlab and Python, as many basic matrix operations are natively available in these environments. In C, the functions are prefixed with the type of data they operate on. For example, fvec_max takes a fvec, or “float vector” array as input. In Matlab, most of the functions are prefixed with yael. In Python, YAEL is available via the ynumpy module, which provides a sufficient context.
2. Main functions

Here we review the implementation of Yael’s main functions, with their names in C. Table 1 gives the Python and Matlab equivalents of these functions.

2.1. k-smallest values query

Yael contains two implementations of the algorithm to find the $k$ smallest values of a table of scalars:

- The quick-select algorithm (a.k.a. Hoare method). It amounts to applying a quick-sort on the table, but without handling the fraction of the table that does not contain the quantile limit.

- The max-heap algorithm, that maintains the set of $k$ smallest values seen so far in a max-heap. The max-heap is a binary tree optimized to perform the operations “remove the largest value” and “insert a new value”, both of which have complexity $\log(k)$.

Figure 1 shows that for smaller quantiles, the max-heap is faster. The function `fvec_k_min` automatically switches between the two versions, depending on the parameter $k$ and number of elements.

2.2. Searching neighbors

Image retrieval requires finding similar elements on large scale. Yael implements exact algorithms to retrieve nearest neighbors. For larger scale, one may resort to indexing structures such as those proposed in the inverted file structure of Yael (see Subsection 2.6).

Exhaustively searching for nearest neighbors requires to compute the distance matrix $D \in \mathbb{R}^{m \times n}$ from the query vectors $X \in \mathbb{R}^{d \times n}$ to the database vectors $Y \in \mathbb{R}^{d \times m}$ (in the Euclidean case). Then, the smallest value in each column $j$ of $D$ is selected: it corresponds to the the nearest neighbor of query $j$. Both computations are performed in blocks to improve cache locality and because $D$ may not fit in memory.

**Euclidean space.** We exploit the equivalence:

$$D = S(X)1_n^T + 1_m^TS(Y) - 2X^TY$$

where $1_n$ is a $n$-vector of ones and $S(X)$ is the vector of squared norms of the columns $X$. The costly operation is the $X^TY$, for which we use the BLAS `sgemm`. When more than one nearest neighbor is required, we use a max-heap over the columns of $D$, as discussed in the previous subsection.

**Hamming space.** To compute Hamming distances, we employ the $\times\oplus$ and SSE4.2 `popcnt` instructions: the Hamming distance between two 64-bit integers is computed in two processor cycles. Hamming distances are typically employed to retrieve all vectors within a given distance to the query. Therefore, unlike in the k-NN case, the list of neighbors must be adjusted during the query.

**Other distances.** The cosine similarity is optimized with matrix multiplication and cache optimization. The $\ell_1$ and $\chi^2$ distances are optimized using specific SSE instructions.

2.3. Principal Component Analysis

Dimensionality reduction with PCA is widely used. However, many implementations are not tractable for a large number of high-dimensional vectors. Yael provides distinct implementations for Matlab and C. The Matlab function `yaelpca` automatically switches between the Gram and the covariance methods [2]. Hereafter, we detail the variants implemented in the C interface, which include both these variants and tricks that limits the memory consumption.

Computing the PCA of a set of vectors $X = [x_1 \ldots x_n] \in \mathbb{R}^{d \times n}$ proceeds as follows.

- centering the data:

$$\bar{X} = X - \bar{x}1_n^T \quad \text{with} \quad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \quad (2)$$

- computing the SVD decomposition of $\bar{X}$: $\bar{X} = U\Sigma V^T$ where $U \in \mathbb{R}^{d \times d}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{d \times n}$ is positive diagonal, padded with zeros on the right or bottom, with elements $[\sigma_1^2, \ldots, \sigma_{\min(d,n)}^2]$. The singular values are sorted: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(d,n)}$. 

![Figure 1. Runtime of the quickselect and max-heap methods to find a quantile in a table of uniformly distributed scalars (one computing core).](image-url)
Given this decomposition, the PCA reduction of a vector \( x \in \mathbb{R}^d \) (not necessarily from the initial \( \{x_1 \ldots x_n\} \)) is \( x' = U^\top (x - \bar{x}) \). Its main property is that, if vectors are drawn from a multidimensional Gaussian, the subvector \( x'_k \) of the \( k \) first components of \( x' \) is as close as possible to \( x' \). Therefore, the PCA is often used to reduce the dimensionality from \( \mathbb{R}^d \) to \( \mathbb{R}^k \).

The only data we need to perform dimensionality reduction is \( \bar{x} \) and the first \( k \) columns of \( U \). Thus, the complete SVD is not necessary. There are two cases:

- Most often, \( n > d \), i.e., we have enough training data. In this case only the eigenvalues \( \sigma_i^2 \) and vectors \( U \) for the empirical covariance matrix \( \bar{X} \bar{X}^\top \) are computed. The cost of computing these eigenvectors is \( 9d^3 \) flops [3, algorithm 8.3.3].

- When \( d > n \), it is more efficient [2] to compute the eigenvectors \( V \) of the Gram matrix \( \bar{X}^\top \bar{X} \). Denoting \((u_i)_{i=1 \ldots d}\) and \((v_j)_{j=1 \ldots n}\) the columns of \( U \) and \( V \) respectively, \( u_i = 1/\sigma_i^2 \bar{X}v_i \) for \( i = 1 \ldots n \). The remaining columns can be obtained by completing the orthonormal basis.

Finding eigenvalues in YAELE uses the Lapack **ssyev** function. When the data matrix \( X \) is too large to fit in RAM, the mean \( \bar{x} \) and covariance \( \bar{X} \bar{X}^\top \) are computed in a single pass over the matrix, using YAELE’s **pca_online** functions.

**Partial PCA.** When the number of required dimensions \( k \ll d \), it is possible to efficiently compute only part of the eigenvalues and eigenvectors. This can be done either by:

- reducing the matrix to tridiagonal form, then only partially diagonalizing this representation (Lapack function **ssyevx**).

- using an iterative Lanczos method [3, section 9.1] to compute the required components. The advantage is that the covariance matrix does not need to be computed, because it is used only to perform matrix-vector operations \( \bar{X}^\top v \) for a given \( v \in \mathbb{R}^d \). YAELE uses the Lapack **ssaupd/ssseupd** functions.

Figure 2 shows the cost of computing a partial PCA. The on-the-fly multiplication is more efficient when \( k \) is small because the covariance matrix is not computed. For larger \( k \), Arpack is faster at first, then Lapack’s tridiagonalization.

These functions are implemented in **fmat_new_pca_part** and **pca_online_complete_part** functions.

### 2.4. k-means Clustering

The cost of the k-means clustering is determined by that of the nearest-neighbor search (see Section 2.2). Despite its simplicity, the performance and stability of k-means depend on some implementation details:

- initialization: YAELE can use a random or the kmeans++-style initialization [1]. We observed, however, that for a large number of centroids, the random initialization is as good, and it is much faster because kmeans++ cannot be parallelized. The user can also provide his own initialization for the centroids.

<table>
<thead>
<tr>
<th>Matlab function</th>
<th>Python function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yael_kmax</td>
<td>kmax</td>
<td>Find the k largest elements of a vector (or set of vectors)</td>
</tr>
<tr>
<td>yael_kmin</td>
<td>kmin</td>
<td>Find the k smallest elements of a vector (or set of vectors)</td>
</tr>
<tr>
<td>yael_cross_distances</td>
<td>cross_distances</td>
<td>Compute all distances between two sets of vectors</td>
</tr>
<tr>
<td>yael_hamming</td>
<td>hamming</td>
<td>Compute all Hamming distances between two sets of packed bitvectors</td>
</tr>
<tr>
<td>yael_nn</td>
<td>nn</td>
<td>Find the nearest neighbors in a set for various distances</td>
</tr>
<tr>
<td>yael_kmeans</td>
<td>kmeans</td>
<td>Learn a k-means quantizer from a set of input vectors</td>
</tr>
<tr>
<td>yael_gmm</td>
<td>gmm_learn</td>
<td>Estimate a GMM from an input set of vectors</td>
</tr>
<tr>
<td>yael_fisher</td>
<td>fisher</td>
<td>Compute the GMM- Fisher vector associated with a given set</td>
</tr>
<tr>
<td>yael_lrf</td>
<td>lrf</td>
<td>Handle (construct, query, etc) an inverted file structure</td>
</tr>
</tbody>
</table>

Table 1. Main functions in the Matlab interfaces of YAELE, and corresponding Python function.
• vanishing clusters: sometimes no more points are assigned to a cluster. This happens even on “well-behaved” datasets, for example when training product quantization [7] vocabularies on very large vectors that may be sparse. In this case, YAEI randomly chooses another cluster and splits it in two, in a way that avoids a new cluster to vanish at the next iteration. This is done by selecting a big cluster associated with centroid c. Two clusters are obtained from it as c − ε and c + ε, where ε is a small random perturbation vector.

• restarting: kmeans computes the solution to an optimization problem using an iterative method that converges to a local minimum. The YAEI kmeans can perform this optimization starting from different initial points and select the best one.

Several functions are available for k-means in the C interface of YAEI, depending on the desired output (average error per cluster, assignment, etc) and initialization options. In Matlab, k-means is called with function yael_kmeans, see help yael_kmeans for possible options. For instance, a simple call is done by

\[
[C, I] = \text{yael_kmeans}(v, \text{‘niter’}, \text{niter});
\]

where C is a matrix containing the k clusters, and I indicates to which cluster each input vector v is assigned.

2.5. Fisher vectors

The YAEI Gaussian Mixture Model [2, Section 2.3.9] estimation is initialized from a kmeans vocabulary of the same size, then expectation-maximization steps are performed. A set of vectors can be aggregated to a Fisher vector [11]. This is particularly useful to generate an image representation from local descriptors.

The two functions are implemented in YAEI’s gmm_learn and gmm_fisher. A toy Matlab package implementing a baseline based on Fisher is available online, see Section 3. A similar Python package is provided for the open-source competition to illustrate YAEI’s capabilities.

2.6. Inverted files

Inverted file systems [14] are a key component of existing large-scale image and video retrieval methods. For image search, they are widely adopted to implement the similarity computation for the methods derived from the seminal Video-Google paper [12], where sparse bag-of-words vectors are to be compared in an efficient way.

Although an inverted file system can be implemented by a sparse matrix-vector multiplication, a devoted implementation can be more efficient. Furthermore, more precise image search systems have been subsequently proposed, such as Hamming Embedding [6], selective match kernel [13] or nearest neighbor search with product quantization [7]. These strategies add some per-descriptor information, given in a form of a short code, e.g., a binary vector.

YAEI implements a generic inverted system capable of storing and exploiting such additional information in the retrieval phase. It can be used with any quantizer, since the quantization is done externally. For instance, in the Matlab interface, one can learn and produce a structure for Hamming Embedding by providing a function handler for the quantizer (exact Euclidean assignment in the example below) when calling the yael_ivf function:

\[
\text{>> quantizer=@yael_nn; } \text{qparams=yael_kmeans(xtrain, k,’niter’,20); } \text{ivfhe=yael_ivf_he (k,nbits,x,quantizer,qparams);}\]

The entry point for all operations is the function yael_ivf: adding new elements, performing queries, saving and loading an inverted file, etc. A full example is given in the subdirectory test/matlab of the YAEI package.

3. Impact of YAEI

This section gives a brief overview of the impact of YAEI, both in terms of packages that depend on it, and with respect to usage statistics measured by the number of downloads. In addition, we mention that about 20 papers or reports refer to YAEI, as indicated by the results of the Google scholar query “gforge inria yaei”.

3.1. Derived software

Several software use YAEI as a core library, in particular online packages that reproduce the results of papers\(^1\). Hereafter, we describe a few packages developed at Inria.

VLAD and Fisher – Image representations. The packages have been released to reproduce the results of the corresponding papers [8, 10]. They are available online on a dedicated webpage\(^2\). The Fisher implementation reproduces the results of the original authors at Xerox Research Center Europe.

Product quantization [7] is a state-of-the-art method for approximate nearest neighbor search in the compressed domain. A commercial version of this algorithm is proposed by Inria. Alternately, an open-source toy package in Matlab is available online\(^3\), and depends on YAEI. Another package implementing approximate search with short codes implements the anti-sparse binarization technique [9] is derived from YAEI\(^4\).

\(^1\)http://people.rennes.inria.fr/Herve.Jegou/software.html
\(^2\)http://lear.inrialpes.fr/src/inria_fisher
\(^3\)http://people.rennes.inria.fr/Herve.Jegou/projects/ann.html
\(^4\)https://gforge.inria.fr/projects/antisparse
4. Conclusion

Yael is a library which offers computationally intensive functions that are critical for large scale image search. Its main purpose is to facilitate the reproducibility of results, thanks to release of packages associated with research papers. Yael offers C, python and Matlab interfaces for Linux and MacOS operating systems. The core functions are heavily optimized with SSE operations and calls to Lapack/Blas matrix libraries. The development process follows the motto “release soon, release often”.

Acknowledgements. The library was first developed in the context of the Quaero program. It is currently supported by ERC VIAMASS no. 336054.

References