2013

Golden Retriever - A Java Based Open Source Image Retrieval Engine

Tsochatzidis, Lazaros T.

ACM

Golden Retriever - A Java Based Open Source Image Retrieval Engine

Lazaros T. Tsochatzidis, Chryssanthi Iakovidou, Savvas A. Chatzichristofis and Yiannis S. Boutalis
Democtitus University of Thrace
Department of Electrical and Computer Engineering
Xanthi, Greece
{lazatsoc,ciakovid,schatzic,ybout}@ee.duth.gr

SUBMITTED TO ACM MULTIMEDIA 2013 OPEN SOURCE SOFTWARE COMPETITION

ABSTRACT

Golden Retriever Image Retrieval Engine (GRire) is an open source light weight Java library developed for Content Based Image Retrieval (CBIR) tasks, employing the Bag of Visual Words (BOVW) model. It provides a complete framework for creating CBIR system including image analysis tools, classifiers, weighting schemes etc., for efficient indexing and retrieval procedures. Its eminent feature is its extensibility, achieved through the open source nature of the library as well as a user-friendly embedded plug-in system.

GRire is available on-line along with install and development documentation on `http://www.grire.net` and on its Google Code page `http://code.google.com/p/grire`. It is distributed either as a Java library or as a standalone Java application, both GPL licensed.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Image Retrieval, Visual Words, Bag-of-Visual-Words, Open Source, Image Search, Image Indexing

1. INTRODUCTION

Content based image retrieval approaches can be mainly classified into two groups based on the types of low level visual features that they employ. The first group consist of methods that use global features while the second one is formed by approaches that employ local features. Additionally, CBIR systems can be separated into two general classes based on the type of the queries they handle. In applied research, CBIR systems are retrieving images with similar visual characteristics to the query image (e.g. if the query is a red image, the retrieved images are also red)[4]. To support these queries, mainly global features are employed. At the same time, many advanced systems are able to recognize the context of the query and to retrieve semantically similar images (e.g. if the query is a red luxurious car, the retrieved images are depicting a Ferrari). In this case, these queries are mainly supported by local features.

Recent CBIR approaches have strongly focused on the combination of global and local features. These approaches are employing the local features in order to produce a holistic, global representation of the image. An example of such approaches is the bag-of-visual-words (BOVW) approach. This BOVW representation is analogous to the bag-of-words representation, a well-known and widely used method in text retrieval, where a document is represented by a set of distinct keywords.

BOVW methods are fast becoming a widely used representation for CBIR, mainly for three reasons: their better retrieval effectiveness over global feature representations, the much better efficiency than local feature representations and the fact that they can be used both, by systems that attempt to identify the content of the query to retrieve images with semantically similar content and by systems designed to retrieve images with similar visual concept to the query.

Figure 1: Golden Retriever Logo.

This paper presents a new open source and extensible Java image retrieval library named Golder Retriever. The scope of the library is to provide solutions on how to integrate CBIR techniques in an easy way, on a wide range of applications. Additionally, GRire provides a testing platform for researchers that could be used to evaluate existing or new CBIR approaches.

Current version of the library is mainly focused on the BOVW approach, integrating a variety of local features, classifiers as well as methods from the field of information retrieval adjusted to meet the image retrieval perspective.

MM'13, October 21–25, 2013, Barcelona, Spain.

2. THE GRIRE PROJECT

The main objective of the project is to help developers create and evaluate their methods in any image database with minimum effort and without needing to concern about the details of the model itself. Furthermore, developers are welcome to integrate custom components, such as feature extractors and descriptors, into the GRire library. Thus, a whole BOVW system can be created and tested adopting the provided implemented weighting schemes and similarity models. Towards this direction, GRire combines an adaptable and easy-to-use plug-in system together with a powerful and efficient indexing and retrieving mechanism.

2.1 Project Structure

The GRire project can be operated in two ways; either as a Java library to be integrated into other applications or through an easily used graphical user interface called GRireFX. These two versions are distributed in the same file available for download, under the packages `org.grire` and `org.grirefx` respectively and they both provide the same functionality and extensibility. Essentially, the GRire library consists of two main parts: the core and the components.

2.2 The Core

The core of GRire implements the main work flow for the indexing and the retrieval procedures using a BOVW architecture. It consists of the classes required by the model and it is considered to be the 'fixed' part of the library. The classes of the core are organized into two different types hereby referred to as *Structures* and *Functions*. *Structures* are objects responsible for handling database's operations such as storing and fetching. On the other hand, *Functions* implement the procedures for generating the model, like indexing and retrieving. Furthermore, *Structures* are usually an 'extra layer' above the *GeneralStorer* which provides more basic structures described in sections 2.3 and 3.2.

2.2.1 Structures

ImagePool An image pool is a collection of images. It pairs a unique id with the absolute path of an image and stores it in a database. It provides all the required methods for adding and deleting an image from the pool.

PoolFeatures This structure is the collection of features extracted from an *ImagePool*.

Codebook The main component of the BOVW model. It is a lexicon of key-features selected from the *PoolFeatures* structure using various ways (clustering etc.).

Index This structure stores the representation (descriptor) of each image.

2.2.2 Functions

Importer This is a simple class that allows batch importing of image from multiple folders.

PoolFeatureExtractor It extracts features from all the images part of an *ImagePool* to generate a *PoolFeature* structure for storing them. These features are extracted using a *FeatureExctractor* component (kindly refer to section 2.3).

ClusteringCodebookFactory This class creates a new lexicon (codebook) by performing a *ClusteringAlgorithm* on the features provided. By default, the *ClusteringCodebookFactory* will create clusters using the 30% of the data but users have the ability to modify the percentage. The training set is stored temporary in the file system.

IndexFactory This function produces the index of the image representations (descriptors) using the *PoolFeatures* extracted earlier. The representation for each image is created according to the supplied *VisualWordDescriptor* component (details in section 2.3).

QueryPerformer This class performs the searching procedure for one or more queries, given as input, generating a TREC formatted file with the results for easy and accurate evaluation.

2.3 The Components

The BOVW model can be assembled by several different components that execute discrete steps of the architecture, thus making it highly customizable. The multiple possible component combinations provide a great variety of experimental setups of the model. Each component allows for independent enhancement attracting researchers/developers from varying scientific fields to explore and extend.

Type	Component Name
`GeneralStorer`	MapDB, See section 3.2
`FeatureExtractor`	SURF [3] SIFT [6] ORB [8] MSER [7] + FREAK [1] CEDD [5] (Bag of CEDDs[1])
`ClusteringAlgorithm`	K-Means SGONG [2]
`SimilarityMeasure`	Euclidean Distance Cosine Similarity
`WeightingScheme`	SMART [9]
`Stemmer`	Euclidean Stemmer

Table 1: A list of the currently implemented Components

These components are interfaces whose implementations are given as arguments to the core classes. A list of the components is shown below while Table 1 briefly presents the implemented components:

1. `GeneralStorer`. It manages the way data are written in the file system, providing simple structures such as maps and sorted maps.

2. `FeatureExtractor`. It is employed from the core for the extraction of local features from an image.

[1]The proposed library has the ability to adjust the information and the advantages derived from the Global low level Features (GF), into the BOVW model. GRire separates the images into a preset number of image blocks and calculates a global descriptor (e.g. CEDD) from each one. Based on this approach, every image is represented with multiple GFs. In the sequel, these global features are considered as visual words.

3. `ClusteringAlgorithm`. This component implements a clustering algorithm used for the creation of the codebook.

4. `SimilarityMeasure`. It is used for the comparison of the representation (descriptor).

5. `VisualWordDescriptor`. This component's task is to form the representation (descriptor) of an image using all the previously created structures.

6. `WeightingScheme`. The weighting scheme used during the retrieval process defines how the final representation (descriptor) of an image will be formed, just before it is compared with other representations.

7. `Stemmer`. Stemmers are the objects that define how a visual word derived from an image will be assigned to a word from the codebook.

2.4 Work-flow

Initially, the images are imported into an `ImagePool` object. Then, the `PoolFeatureExtractor` extracts the local features from the imported images, using a `FeatureExtractor` component and stores them in a `PoolFeatures` structure. In the sequel, the `IndexFactory` creates the representation (descriptor) for each image using a `VisualWordDescriptor` component, according to a `Codebook` structure. The results are stored it in the `Index`. The codebook was created by the `CluteringCodebookFactory` class in advance, using a `ClusteringAlgorithm` component on `PoolFeatures`. The system is now ready to perform any retrieval task using the `QueryPerformer` together with the `WeightingScheme` and `SimilarityMeasure` components.

3. TECHNICAL DETAILS

3.1 Extending GRire

GRire has been designed with particular attention to be highly customizable giving the opportunity to developers / researchers to implement their own methods and integrate them in the framework. This is achieved through the plug-in system which adopts the open source *jspf* [2]. Developers can easily create a plug-in without any constraints on the number of the needed parameters so as to implement one of the components described earlier. For the users' convenience a set of common parameters has been predefined to be recognizable by the framework. The plug-in interacts with GRire notifying it about the number and the type of the parameters and the program dynamically prompts the user for the right input method. Every plug-in has the following functionality:

- It implements only one component of GRire.
- It may require input parameters.
- It may require a setup process manually called by the user before it can be executed.
- Additional parameters for the aforementioned setup process may be also required.
- Default initialization values may be available for all or some of the parameters.

[2] `https://code.google.com/p/jspf/`

3.2 Storage

The GRire library employs the very powerful and promising *MapDB*[3] that supports fast and efficient storage and serialization of the data. This embedded database provides maps (Hash, Sorted and Multi-maps) as basic structures that are used to form all other more complex needed structures (Codebook etc.). *MapDB* offers its own serialization, compression and cache memory system while it is portable and allows processing of large amount of data, with minimum overhead in memory.

3.3 Compiling and Installation

It is decided that the components of GRire will be distributed separately from its core, grouped in plug-in packs. Every component may have its own dependencies, which may be external libraries or even programs. The *git* repository of the project includes the source code of the core along with the source code of every component pack that will be available as plug-in. A universal *ant* configuration file accompanies every module (core and packs) as well as every plug-in pack.

After obtaining the core *jar* file, the user can execute it to launch the GRireFX or import it as an external library to another Java application. The components can be used integrated with the following ways:

- If GRire operates as a Java library, then the plug-ins can be also imported to the application to be used like any other class.
- If GRire operates through its GUI then a text file named 'plugins' is required to be present in the same folder as the *jar* file. This text file contains the paths to the *jar* files of the plug-in packs.

4. GRIREFX

For users that do not wish to create their own application and just want to test their component of the BOVW model, the ideal solution is the graphical user interface of GRire. It is designed not to insert any limitation compared to the GRire library version, retaining its customizability by allowing enrichments in the form plug-in. In figure 2 there is a screen shot of GRireFX while the user is setting up a multi-query retrieval task. The results of the procedure is a TREC formated file.

As mentioned before, GRireFX automatically recognizes the number and the types of the parameters, as well as their default values (if they exist), and dynamically requests from the user the corresponding input. Figure 3 illustrates the dynamic tree of parameters that grows according to every component's needs while the user makes interacts with the interface.

5. PERFORMANCE

Indicative experiments have been performed to evaluate the performance of the GRire library. It is worth noting that, the proposed project is a framework for implementing and integrating various features with different computational costs and complexity, so the time needed for the indexing as well as for the retrieval procedure greatly varies and depends on the components that will be used (features,

[3] `https://github.com/jankotek/MapDB`

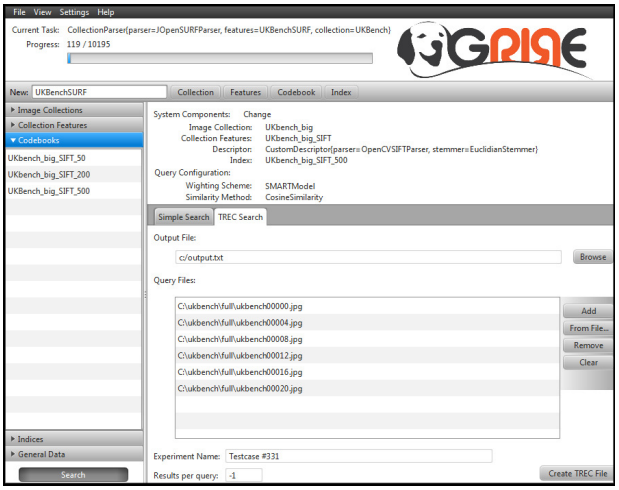

Figure 2: GRireFX performing a multi-query experiment.

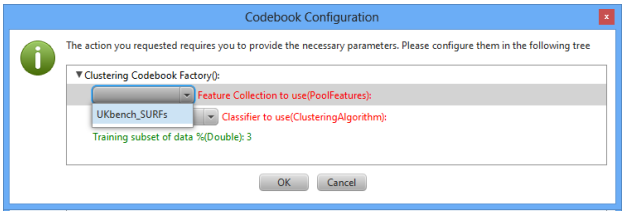

Figure 3: GRireFX dynamically adds leaves to the tree of parameters.

classifiers, codebook size, weighting scheme, database size, etc). Table 2 presents the time (in minutes) elapsed for two different experiments that have been performed.

	SURF	SIFT
Feature Extraction	109	163
Codebook Generation (K-Means)	73	158
Descriptor Calculation	20	26
Retrieval Procedure (100 queries)	<2	<2

Table 2: Results of the time needed for each process measured in minutes.

For the first experiment, the SURF(64) features were used as the local features, while for the second one, the SIFT(128) features were employed. In both cases, experiments conducted on the *UK-BENCH* [7] image dataset (10200 images), using a codebook of 1024 visual words. For the retrieval procedure, we executed a multi-query experiment (100 queries linear search) using the tf-idf weighting scheme and cosine similarity measure. Experiments carried out on a Core 2 Quad Q6600 CPU with 4GB of RAM system. The storage needs for the created databases reached a total of 2.89 GB and 5.48 GB respectively.

6. CONCLUSIONS

This paper presents a new framework for CBIR based on the BOVW approach, that can be used in a wide range of applications. The open source nature of the proposed library allows and encourages researches to extend GRire by implementing more components, methods and features. It's worth noting that apart from a powerful and fast indexing and retrieving mechanism, GRire additionally uses an extremely easy to use plug-in system. The GRire library is provided under GPL license and is available online along with install and development documentation on `http://www.grire.net` and on its Google Code page `http://code.google.com/p/grire`

7. ACKNOWLEDGMENTS

This research has been co-financed by the European Union (European Social Fund-ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF)- Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

8. REFERENCES

[1] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *CVPR*, pages 510–517, 2012.

[2] Antonios Atsalakis and Nikos Papamarkos. Color reduction and estimation of the number of dominant colors by using a self-growing and self-organized neural gas. *Eng. Appl. of AI*, 19(7):769–786, 2006.

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc J. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[4] S.A. Chatzichristofis, C. Iakovidou, Y. Boutalis, and O. Marques. Co.vi.wo.: Color visual words based on non-predefined size codebooks. *Cybernetics, IEEE Transactions on*, 43(1):192–205, 2013.

[5] Savvas A. Chatzichristofis and Yiannis S. Boutalis. Cedd: Color and edge directivity descriptor: A compact descriptor for image indexing and retrieval. In *ICVS*, pages 312–322, 2008.

[6] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[7] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, volume 5, pages 2161–2168, 2006.

[8] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[9] G. Salton. *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.