This project that has received funding from the European Union's Horizon 2020 - Research and Innovation Framework Programme, H2020-FCT-2015, under grant agreement no 700381.

Analysis System for GAthered Raw Data

ASGARD

Instrument: Research and Innovation Action proposal

Thematic Priority: FCT-1-2015

D10.3. Large-scale multimedia indexing tools

Deliverable number	D10.3	
Version:	1.0	
Delivery date:	30 April 2019	
Dissemination level:	OTHER	
Classification level:	PUBLIC	
Status	DRAFT	
Nature:	OTHER	
Main author(s):	Apostolos Axenopoulos, Michalis Lazaridis	CERTH
Contributor(s):		

DOCUMENT CONTROL

Version	Date	Author(s)	Change(s)
0.1	20/03/2019	Apostolos Axenopoulos (CERTH)	Structure, TOC.
0.2	20/03/2019	Apostolos Axenopoulos, Michalis Lazaridis (CERTH)	First version
0.3	15/04/2019	Apostolos Axenopoulos, Michalis Lazaridis (CERTH)	Ready for internal review
0.4	23/04/2019	Stephan Raaijmakers (TNO), Ernesto La Mattina (ENG)	Internal Reviews
0.5	25/4/2019	Romaios Bratskas (ADITESS)	Quality review of the deliverable
1.0	26/4/2019	Apostolos Axenopoulos, Michalis Lazaridis (CERTH)	Final version

DISCLAIMER

Table of Contents

Tables

Figures

1.Introduction

1.1. Overview

The DoA describes this deliverable as:

D10.3 – Large-scale multimedia indexing tools. [Month 32, Type: other, Dissemination level: PU]

The deliverable describes the work of subtask 10.2.2 of WP10, which aims to provide tools for large-scale media indexing, search and retrieval. The work focuses on search-by-example tasks, i.e. to search for relevant image, audio or video by using as query a sample image, audio or video, respectively. The typical search procedure involves a vector comparison between the descriptor of the query sample with the descriptors of all items within the database, using e.g. classical distance measures (Euclidean, Manhattan, etc.). However, as the size of available data to search increases dramatically, the time for the above search process becomes prohibitive. Therefore, more efficient approaches for media indexing are required. The contribution of this task is twofold: i) it provides a state-of-the-art method for large-scale image indexing and search and ii) it proposes a new add-on for MongoDB to support vector-based indexing and search, applicable to all media types (e.g. image, audio, video).

1.2. Relation to other deliverables

This deliverable is related to the following other ASGARD deliverables:

Receives inputs from:

Deliv. #	Deliverable title	How the two deliverables are related
D5.2	Datasets Acquisition and/or creation	It identifies and therefore provides all the available, relevant datasets alongside with the internal catalogue for various data sources and metadata.
D5.3	Data Filtering and Interoperability	Provides to this deliverable the standards, data access and format alongside with the organisational interface.
D5.4	Metadata Extraction and Augmentation	Identifies relevant metadata attributes to enhance data retrieval capabilities and improve the description of the datasets provided to D10.3.
D6.1	Text Analysis	Textual descriptions extracted from text analysis can be used as input for large-scale indexing.
D6.2	Audio Analysis	Audio descriptions extracted from audio analysis can be used as input for large-scale indexing.
D6.3	Image Analysis	Image descriptions extracted from image analysis can be used as input for large-scale indexing.
D6.4	Video Analysis	Video descriptions extracted from video analysis can be used as input for large-scale indexing.
D9.1	Orchestration framework definition and	It specifies all the characteristics and requirements

	set-up.	to be met from the tool presented in D10.3.

Table 1 – Relation to other deliverables – receives inputs from

Provides outputs to:

Deliv. #	Deliverable title	How the two deliverables are related
D7.5	Visual analytics framework and techniques	Results from large-scale retrieval can be exploited by visual analytics for efficient presentation/visualisation.
D10.4	Continuous Integration & Common Infrastructure, Tools and Resources	The output of D10.3 will be useful for D10.4 as it will serve as a guideline to validate the developed system, alongside with the evaluation of the designed tools. The goal is to identify if the overall large-scale indexing approach complies with the defined requirements.

Table 2 – Relation to other deliverables – provides outputs to

1.3. Structure of the deliverable

This document includes the following sections:

- Section 1: The current introductory section. This section describes additionally all the links and dependencies with other deliverables.
- Section 2: This is the main section, where the Large-scale media indexing framework is described, which includes the implemented architecture, tools and their setup.
- Section 3: In this section, conclusions, limitations and future work are presented.

1.4. Where to find more about this deliverable

Title	Type	SW release	Brief description	Where can it be found
Binary hasher	Source code, documentation and container Image	V0.9	Source code, documentation and complete image of the Binary hasher tool that extracts binary hashes out of images and video content.	ASGARD Gitlab Instance [https://ci.tno.nl/gitlab/ASGARD/demonstrator-wp10-t2.2_image_hashing]
Mongo binary hash indexing	Source code, documentation and container Image	V0.9	Source code, documentation and complete image of the MongoDB branch that supports binary hash indexing, search and retrieval.	ASGARD Gitlab Instance [https://ci.tno.nl/gitlab/ASGARD/demonstrator-wp10-t2.2_Mongo_binary_hash_indexing]

Table 3 – References for the Container

2.Framework Description

2.1. Workflow description

The workflow of the proposed approach is depicted in Figure 1. During the descriptor extraction step, the input file is given to the Descriptor extractor. The input can be an image, video file, audio file or text, i.e. any piece of media from which a multidimensional descriptor vector can be extracted. The descriptor extractor extracts automatically low-level features from the file (e.g. shape, texture, colour characteristics from visual content, audio-based features from audio content or text-based features from texts) represented as multidimensional vectors. In case the descriptor vector values are real numbers, an additional step is required to convert the real numbers to binary hashes. In this task, we have implemented a tool that extracts binary descriptor vectors from images and is described in the following subsection. However, this tool can be substituted by any image (or audio/video/text) descriptor extractor, provided that it conforms with the development guidelines of ASGARD and that the output descriptor is a binary hash.

The next step, MongoDB with vector indexing support, has two operation modes:

- Ingestion: takes as input the binary hash of an image (from the previous step) and related metadata and stores them to the index.
- Search and retrieval: the image or binary hash are given as input and the tool returns a list of the most visually similar image records.

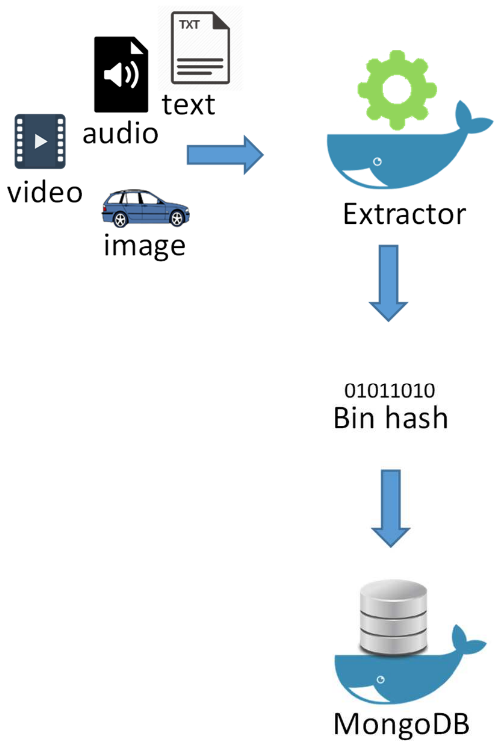

Figure 1 – Generic workflow

2.2. Extraction of Binary Vector

Given an image as input, the Descriptor Extractor computes the visual descriptors. Since the next step of MongoDB vector indexing requires the descriptors to be binary hashes, an additional step of converting the descriptors into binary hashes might be required if they are real numbers.

Several methodologies for extracting low-level image descriptors have been proposed in the literature, based either on hand-crafted features or on machine learning, describing either the entire image or local regions and specific objects within an image. In the context of this task, some of the most widely-used image descriptors have been tested. A report on these methods is out of the scope of this deliverable. The descriptor extractor that has been eventually implemented is based on a work presented in CVPR2015[1]. The reasons for selecting this approach are i) deep-learning-based image features pre-trained on ImageNet have been proven to be more discriminative and ii) the proposed approach outputs directly binary vectors as descriptors, thus, the additional step to create the binary hash is avoided.

The method is outlined in Figure 2. *Module 1* involves the supervised pre-training on the large-scale ImageNet. This step learns rich mid-level image representations. *Module 2* focuses on fine-tuning the network with the latent layer to simultaneously learn domain-specific feature representation and a set of hash-like function. Finally, *Module 3* performs the search-by-example image task, where images similar to the query are retrieved following a hierarchical search process.

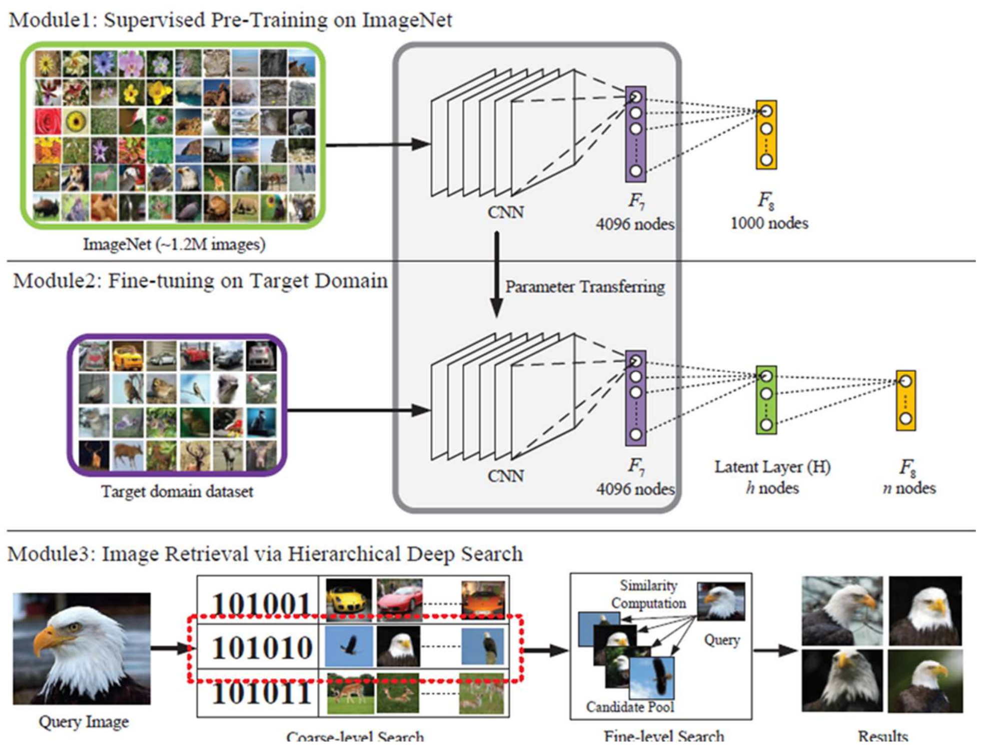

Figure 2 – Binary hash extraction

The output of Module 2 is a binary hash of length 48, which corresponds to the number of nodes *h* of the

[1] Deep Learning of Binary Hash Codes for Fast Image Retrieval K. Lin, H.-F. Yang, J.-H. Hsiao, C.-S. Chen CVPR Workshop (CVPRW) on Deep Learning in Computer Vision, DeepVision 2015, June 2015.

latent layer. Two different binary code lengths have been tested, *h*=48 and *h*=128. The size of the code length is not always proportional to accuracy; in some cases, a larger binary hash is required to capture higher details, while in some cases, more nodes of the latent layer may cause overfitting. During image retrieval, the coarse-level search selects the subset of images from the database with a Hamming distance to the query hash lower than a threshold, while the fine-level search re-ranks the subset of remaining images from the coarse step. The supported image formats are .png, .jp(e)g, and .bmp. The concept is extended to support also video frames. The supported video file formats are .mp4, .avi, .wmv and .mov.

The tool documentation includes detailed instructions on installing, using and testing the tool.

2.3. MongoDB vector indexing

Normally, fields that can be indexed in MongoDB are either single numbers, text or geospatial data (long, lat). There is no support for indexing a descriptor vector, coming e.g. from an image. For addressing this need, we use a two-step procedure: i) we convert descriptor vectors to binary hashes, which are then stored in a specific field of the MongoDB document; ii) we index this field, using a specific index type, so that the documents are later searchable based on the binary hash (and thus the descriptor vector) similarity. The first step is covered by the method described in the previous subsection, while the second is addressed by the MongoDB vector indexing, that has been introduced in the context of this subtask (T10.2.2).

The MongoDB vector indexing introduces a new index type, capable of indexing binary hashes and comparing documents based on these hashes. The current master version is built on top of the r3.4.7 version of mongoDB. The distance metric used for the binary hashing index is the Hamming distance. For binary strings *a* and *b* the Hamming distance is equal to the number of ones in *a XOR b*.

The binary hashes are formatted as strings and can be placed in a document field. E.g.

```
{
        "_id": ObjectId("5aec22e800b881198b8e682a"),
        "bhash": "101011110011011101000101111010000100111011110010",
        ...
}
```

The length of the binary hash does not need to be specified. This tool, if combined with a binary hash extractor (like the Binary Hashing tool), can be used as an out-of-the-box search engine.

The tool accepts requests exactly as a MongoDB instance, either through the standard port 27017 (same docker network) or through the port 9010 (outside world). Any MongoDB API, driver or client, including a MongoDB shell, can be used.

The tool documentation includes detailed instructions on installing, using and testing the tool.

3.Conclusions

3.1. Summary

The current work introduces a compact, minimal configuration, out-of-the-box framework for indexing, search and retrieval of large amounts of media content.

The framework consists of two main parts:

- The Binary Hasher extracts binary hashes out of a media file.
- The MongoDB branch supports binary hash indexing, search and retrieval.

Two working modes can be distinguished:

- During the indexing phase, the user provides an image path, a video path or an image folder path. The tool reads the files, extracts the binary hashes and creates records in the MongoDB.
- During the search phase, the user provides an image path and the tool performs similarity search inside the MongoDB. The results are returned to the caller.

All interfaces are documented in detail in the ASGARD Gitlab, at the locations provided in Table 3.

3.2. Evaluation

The proposed framework is useful as an off-the-shelf, horizontally scalable, content-based search engine. The combination of the docker service replication mechanism and the MongoDB replication and sharding capabilities ensure data redundancy, availability and distribution with high throughput.

A series of tests has been conducted in order to evaluate the performance of the framework. The tests have been conducted on a single mongo instance, without replica sets or sharding, so that the core performance of the concept can be evaluated. The results of the tests are presented in the following table.

Number of images (millions)	**Avg. search time (seconds)**
1	0.8
2	1.3
3	1.9
4	2.4
5	2.9

Table 4 – Search times

We can see that the tool has a nearly linear (slightly below) behaviour to the number of images. This can be significantly improved by using MongoDB's replication/sharding mechanisms.

The proposed binary hash has been tested in three different image datasets, outperforming similar state-of-the-art approaches. For quality evaluation results, please refer to the scientific paper, presented in Section 2.2.

3.3. Limitations and future work

The current implementation of the binary hasher covers image data. For covering also the cases of video, audio and text data, different binary hashers need to be implemented.

There are currently two limitations:

- The new binary hash index cannot be used in the same query with a geospatial index[i].
- The WiredTiger storage engine is not supported. Only the MMAPV1 storage engine[ii] is supported.

Both limitations are expected to be lifted in a future release.

---

[i] https://docs.mongodb.com/manual/geospatial-queries/

[ii] https://docs.mongodb.com/manual/core/storage-engines/