www.forgetit-project.eu

ForgetIT

Concise Preservation by Combining Managed Forgetting and Contextualized Remembering

Grant Agreement No. 600826

Deliverable D4.2

Work-package	WP4: Information Consolidation and Concentration
Deliverable	D4.2: Information analysis, consolidation and concentration techniques, and evaluation - First release
Deliverable Leader	Vasileios Mezaris, Vassilios Solachidis, Olga Papadopoulou
Quality Assessor	Walter Allasia
Estimation of PM spent	20
Dissemination level	PU
Delivery date in Annex I	31-01-2014 (M12)
Actual delivery date	4-02-2014 (M12)
Revisions	0
Status	Final
Keywords:	text summarization, term cloud, text condensation, word redundancy removal, semantic enrichment, feature extraction, concept detection, image quality, face detection, image clustering, image summarization

Disclaimer

List of Authors

Olga Papadopoulou / CERTH
Vasileios Mezaris / CERTH
Vassilios Solachidis / CERTH
Anastasia Ioannidou / CERTH
Bahaa Beih Eldesouky / DFKI
Heiko Maus / DFKI
Mark A. Greenwood / USFD

Contents

Executive summary

Preservation and forgetting processes can be assisted by textual and multimedia content analysis and condensation methods. Such methods are necessary for supporting a gradual forgetting approach, where content is preserved at varying levels of detail with the passage of time and the consequent changes in its importance.

In this document we present the first release of the ForgetIT techniques for textual and multimedia information analysis, consolidation and condensation. Additionally, preliminary results of the evaluation of the developed techniques are reported. The theories adopted as the foundation of the presented methods have been studied and analysed in deliverable D4.1 [1].

Specifically, the problems addressed and the methods that are presented in this deliverable, in response to these problems and in accordance with the WP4 objectives, are:

- "Text summarization", for creating a summary of a single document or of a collection of documents.
- "Text condensation", for performing linguistic processing for document length reduction by removing or replacing potentially redundant words without changing the meaning of it.
- "Semantic text composition", which facilitates providing context for the text being composed.
- "Feature extraction and concept detection in image collections", which extracts a vector representation for each image and then utilizes machine learning techniques for quantifying the relation between the image and a set of semantic concepts.
- "Image quality assessment", which attempts to quantify the visual quality of images.
- "Face detection for clustering", which aims to detect faces in image collections.
- "Image clustering for summarization", which groups similar images and condensates the initial image collection.

First software implementations for all these methods have been developed and are presented as part of this deliverable.

Glossary

List of important terms and acronyms presented in the document:

AP	Affinity Propagation
BIQI	Blind Image Quality Index
BoMs	Bag-of-Models
BoW	Bag-of-Word
BRISQUE	Blind/Referenceless Image Spatial Quality Evaluator
DCT	Discrete Cosine Transform
DoC	Degree of Confidence
farthest first	Farthest First Traversal Algorithm
GATE	General Architecture for Text Engineering
GGD	Gaussian Distribution
GPU	Graphic Processing Unit
hier-comp	Hierarchical clustering using complete linkage
hier-single	Hierarchical clustering using single linkage
IQA	Image Quality Assessment
LOD	Linked Open Data
LSVM	Linear Support Vector Machine
MOS	Mean Opinion Score
NER	Named Entity Recognition
NLP	Natural Language Processing
NMI	Normalized Mutual Information
NSS	Natural Scene Statistics
PAM	Partitioning Around Medoids
PIMO	Personal Information Model
PR	Processing Resource
REST	REpresentational State Transfer
RMS	Route Mean Square
seed	Semantic Editor
SaaS	Software as a Service
SIFT	Scale-Invariant Feature Transform
SIN	Semantic Indexing
SURF	Speeded Up Robust Features
SVM	Support Vector Machine
xinfAP	Extended Inferred Average Precision
XML	eXtensible Markup Language

1 Introduction

Either for personal or organizational use [2], the amount of media items produced every day is huge. In the case of organizational use, taking into account the information located in the web sites and in the documents and media produced in institutions and large medium enterprises we realize that the amount of text and media that is created, continuously grows at increasing pace. Similarly, in the personal case scenario, digital cameras, mobile phone cameras, web cams, etc. are frequently used, giving the opportunity to the people to capture events important for them, like weddings, birthday parties, a trip, etc., as well as everyday situations like a coffee break with friends, a walk to the beach, etc. All these captured items need storage space to be kept. Nevertheless, not all of them are important and should be preserved along time (e.g. some documents become outdated at some point, or images may exist that are just duplicates or of low quality). Thus, in order to organize, retrieve, process and finally preserve these media items efficiently and effectively, textual and visual analysis techniques are essential. For example text summarization and condensation or image collection quality and concept detection can give the opportunity to manipulate the collection in a human understandable way and distinguish important from unimportant items.

This deliverable presents the first release of the ForgetIT text and visual information analysis techniques for condensation and summarization. Based on the state-of-the-art and requirement analysis that was reported in deliverable D4.1 [1], a first set of ForgetIT analysis techniques were developed and implemented as ForgetIT components, and preliminary performance evaluation of them was also carried out.

The following sections are structured beginning with a problem statement (which includes a short review of the current state of the art). Afterwards we introduce the approach that we developed in ForgetIT and present the results of preliminary experiments conducted for evaluation. We then present the specifications, access and usage information for the software implementations of the developed techniques, and finally we give an outlook on the next steps of our work. All the methods that are described will eventually be part of the middleware component (D8.1 [3]) of the Preserve-or-Forget platform that is being developed in the project.

We start in Section 2 with Text Summarization, in which two components are presented, content and term detection. Content detection detects useful text sections and is used as a first step to summarization. Term detection can extract representative weighted terms from documents, which can be used in order to provide a corpus summary.

Section 3 presents Text Condensation, which is applied in the cases that summarization is either unnecessary, illegal (for regulatory reasons), or simply unwanted. Text Condensation performs linguistic processing for the purpose of document length reduction by removing or replacing potentially redundant words without changing the meaning of it or removing any details.

Textual information processing is concluded in Section 4, where a Semantic Text Compo-

sition tool is presented. This tool facilitates providing context for the text being composed, by exploiting natural language processing tools and knowledge from Linked Open Data (LOD). This procedure improves the process of text composition and brings semantics into the text at composition time.

We proceed in Section 5 to present our visual information analysis techniques, starting with feature extraction and concept detection from images. An overall component is created consisting of two processes. The first one, feature extraction, extracts a vector representation for each image of a collection. The second one, concept detection, uses the extracted vector representations to assess and quantify the relations between an image and a set of semantic concepts.

We continue in Section 6 with a technique for image quality assessment. Several visual quality measures (blur, contrast, darkness and noise) are examined and their output is merged, so as to produce a global quantitative assessment of the images' visual quality.

Since the presence of humans is one of the most important semantic elements of an image in many cases, face detection and subsequent clustering are important aspects of visual information processing and summarization. In Section 7, a face detection method is proposed which aims to decrease as much as possible the rate of false face detections.

In Section 8 clustering approaches are used in order to identify groups of similar images and use them for condensating the initial collection. Our evaluation experiments focused on six clustering algorithms and three different types of data features, based on which a combination of algorithm and features is chosen for implementing the initial ForgetIT approach to image clustering for summarization.

The deliverable concludes in Section 9 with a general summary of what was presented in the preceding sections and brief general concluding remarks on the future plans of ForgetIT in the area of multimedia analysis and consolidation. As aforementioned, the software components described will be integrated within the Preserve-or-Forget platform according to a Software as a Service (SaaS) approach, with distributed services connected with REST interfaces.

2 Text Summarization

2.1 Problem statement

Traditional textual summarization is used to provide a summary of a single document or of a collection of documents. This section describes a number of possible solutions which, while primarily developed within other projects, were adopted for use by ForgetIT and are being further developed in accordance with the ForgetIT needs. Future versions of these components will be further adapted to the more detailed requirements posed by the ForgetIT use cases (described in WP9 and WP10 [2]).

2.2 ForgetIT approach

GATE [4], a General Architecture for Text Engineering, has been continuously developed over a period of almost 20 years at the University of Sheffield and provides a large number of components (known as processing resources) which can be combined in many different ways to perform text analysis of one form or another. Some of the resulting applications can be used for summarization or for pre-processing to enable a more tailored approach to summarization. All of these components are available for use within the project, but two specific components are likely to be of immediate use and will be incorporated within the ForgetIT framework at the earliest opportunity; content and term detection.

Content Detection, available via the BoilerPipe plugin [5, 6], is a useful summarization tool. This component analyzes a document to determine which sections are useful in terms of content and which are extraneous in some way. This is most useful when dealing with web pages which often (if not always) contain menus, sidebars, etc. which are not relevant to the content. Identifying and removing these sections is an important first step to summarization of any form as the irrelevant material could easily bias a generated summary. We see this component (or a more complex application containing this component) being of specific interest to the personal use case (see WP9 [2]) where web pages stored in the PIMO could be processed to remove the extraneous material before archiving.

One area in which summarization techniques can provide a clear benefit will be in providing easy to absorb overviews of archived documents. While a collection of documents will be archived along with a description of the contents, this will not always convey the full range of information the package contains in the same way that some form of multi-document summary might. We are currently approaching this problem by using TermRaider [7, 8], to extract representative, weighted terms (words, entities etc.) from documents. These weighted terms can then be used in a number of ways to provide a corpus summary. We are currently using them to produce term clouds. A term cloud is a form of weighted list, which allows text to be visualized by equating font size with importance (for a comprehensive overview see [9]). In their simplest form term clouds can be

Figure 1: Example Term Cloud

generated from the raw tokens present in a document, however, it is usually more useful to perform some level of filtering to select appropriate terms from a corpus which can then be visualized and as previously mentioned, in ForgetIT this filtering is carried out using TermRaider. An example term cloud produced in this way from a random sample of 450 tweets from the BBC, the Guardian, and CNN can be seen in Figure 1.

Whilst this approach to summarization may appear simplistic, in that it does not attempt to generate a coherent textual summary, such summaries are easy to understand, even with just a quick glance, and as such should be a useful addition to the use case tools, especially when a user is browsing the archive.

2.3 Software implementation

All the components discussed within this section are already available as part of the standard release of GATE. Some, however, have seen recent development which has not yet been officially released and is only available via a nightly build of GATE [10]. A summary of the technical details of the software is shown in Table 1.

Table 1: Software technical details

Functional description	GATE Based Text Analysis
Input	Text Document
Output	Text Document or Textual Information
Language/technologies	Java, GATE
Hardware Requirements	Any HW with Java support and sufficient RAM
OS Requirements	Any OS with Java support

2.4 Conclusions and future work

In this section we have described a couple of components that can be used to either produce some form of textual summary or that can aid in the production of a summary. These components were originally developed mostly outside the scope of ForgetIT, but work currently under way within the project aims to tailor these components for a better fit with the project goals. This includes tighter integration with the contextualization components being developed in WP6; for example using contextualization to merge multiple terms produced by TermRaider which reference the same entity, and by linking terms to the contextualization information the summary becomes a way of exploring the wider archive as well as summarizing a single archive package.

These improved components will be reported in more detail in deliverables D4.3 and D6.3, but are likely to be integrated within the ForgetIT framework well in advance of the delivery of these documents.

3 Text Condensation

3.1 Problem statement

There are many situations in which time and effort have been expended on producing a document and in these cases summarization is either unnecessary, illegal (for regulatory reasons), or simply unwanted.In those cases where summarization is possible but unwanted, it may still, however, be beneficial to long term preservation if subtle reductions in length can be performed without altering the meaning in anyway; even the reduction of a few characters per documents quickly adds up, and while disk space may be cheap it is not free, especially at larger volumes.

3.2 ForgetIT approach

The approach we have taken to this problem within ForgetIT is to use linguistic processing to determine potentially redundant words and phrases, which can be removed or replaced, without changing the meaning of the document, but which reduce its length.

The exact details of the language constructs that can be simplified were discussed in detail in deliverable D4.1 [1], but some examples of the condensation that can be applied include:

- "Any *particular type of* dessert is fine with me." becomes "Any type of dessert is fine with me." saving ten characters.
- "During that time period, many buyers preferred cars that were pink *in colour* and shiny *in appearance*." becomes "During that time period, many buyers preferred cars that were pink and shiny." saving twenty-three characters.
- "The function of this department is *the* collect*ion of* accounts." becomes "The function of this department is to collect accounts." saving seven characters.

3.3 Software implementation

This work has been implemented as a GATE [4] processing resource (PR) and uses WordNet [11] as its main linguistic resource. As a GATE PR, this component can be used on its own or embedded within a larger application (named entity extraction, contextualization, etc.) and can be obtained from `http://downloads.gate.ac.uk/forgetit/LinguisticSimplifier.zip`.

The GATE API makes it trivial to make an application available as RESTful web service or to embed the application within other software. A summary of the technical details of the software is shown in Table 2.

Table 2: Software technical details

Functional description	Linguistically Based Condensation
Input	Text Document
Output	Text Document
Language/technologies	Java, GATE, WordNet
Hardware Requirements	Any HW with Java support and sufficient RAM
OS Requirements	Any OS with Java support

3.4 Conclusions and future work

Currently the software accompanying this deliverable includes a simple GATE pipeline to allow the PR to be used within GATE. Integration with the ForgetIT framework (either supplying a REST service or embedding within a use case tool) is slated to take place within the next few months as the platform matures.

4 Semantic Text Composition

4.1 Problem statement

Composing natural language text is a task encountered in countless use-cases in real life. Examples include journalists writing news articles, researchers authoring scientific papers, persons writing their diaries, users composing posts on social media websites, customers writing reviews for products, ... etc.

Modern text editing and composition tools offer a variety of features that aim at improving productivity and usability - to name a couple of aspects - of the overall experience of composing natural language text. However, nearly all of those features are based on the ability to understand the syntax of the text rather than its semantics. Exploiting knowledge about the content of the text has potential benefit for the task of text composition and the goals of ForgetIT. By exploiting natural language processing tools, extracting knowledge existing in the text and combining it with knowledge from Linked Open Data (LOD)[12], the process of text composition could be substantially improved and semantics could be brought into the text already at composition time. Those improvements utilize the semantics of the text being composed and bear the potential of enriching user's knowledge at the same time. The term semantic text composition refers to the enrichment of text composition by making use of the semantic information extracted from the text as well as knowledge about the content of the text from sources such as Linked Open Data, or personal knowledge models.

4.2 ForgetIT approach

Textual information condensation and preservation will benefit from semantic text composition in many ways. Semantic text composition facilitates providing context for the text being composed. While a user types in the text, contextual information about the important entities in the text and the relations among them is offered in an interactive way. This also allows for annotating the text at the time of composition and reduces the need for manually adding annotations. Furthermore, annotating the text at such an early stage increases the quality of the information used for annotating it since the author has the opportunity to revise, edit, or reject suggested annotations. It also reduces or even eliminates the need for a later stage of annotation.

In ForgetIT deliverable D9.1 [2], we identified several scenarios in personal preservation where persons are likely to write texts, e.g., taking notes to prepare and during trips, to write down short stories when organizing family photos, or for notes concerning their hobby. The types of texts range from short sentences to bullet points up to several paragraphs. The texts use either already existing entities from the Personal Information Model (PIMO, see deliverable D9.1 [2]) or introduce new ones. To support users, those entities can be identified using Linked Open Data sources such as DBPedia[13], proposed to

them as some kind of explanation of the entities and in a second step, also as annotations of the text. This shall foster the annotations of things in the PIMO and reduce manual effort. Likewise, the contextualization process in WP6 will get additional and more precise textual content (because entities are already linked by the users) to work with.

Therefore, the goal is to provide a text editor component for semantic enrichment of texts written by users. In our work on semantic text composition, we developed *seed* - short for *Semantic Editor*. It is an HTML based rich-text editor supported with natural language capabilities and integrated with personal as well as general knowledge sources.

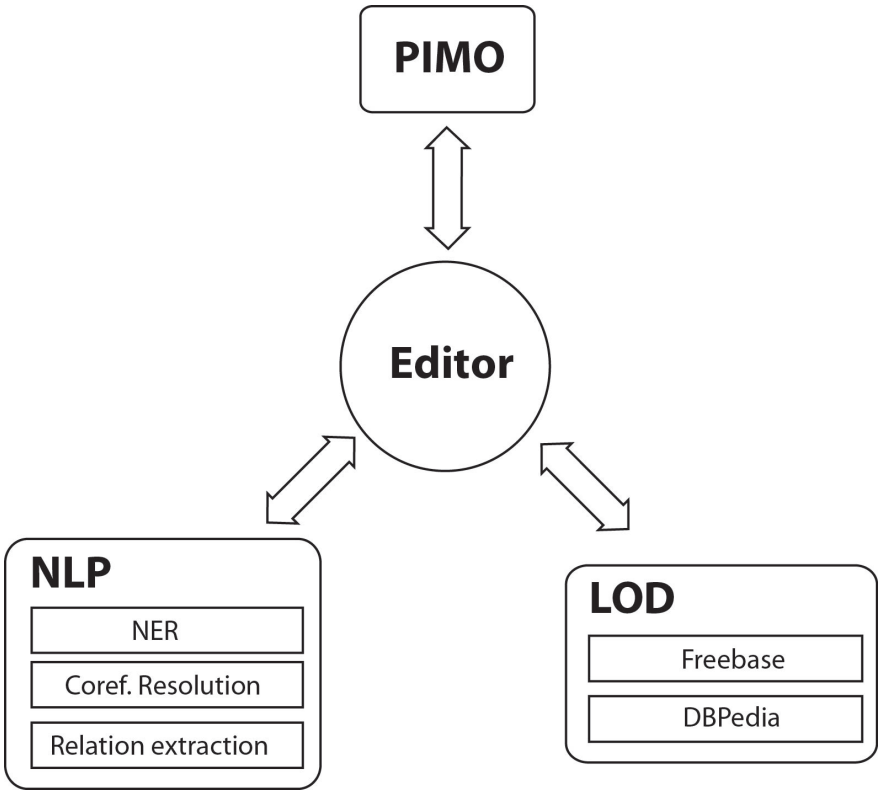

Figure 2: *seed* Components

In Figure 2, *seed* components are shown. From an NLP (Natural Language Processing) perspective, *seed* builds on multiple NLP tools making it capable of performing high level tasks such as named entity recognition, relation extraction, and coreference resolution in addition to a variety of basic NLP tasks. As for the knowledge support in *seed*, we distinguish between two types: personal and general knowledge. Personal knowledge refers to structured information about concepts from the user's point of view (i.e., according to the user's mental model). In this context, PIMO is the personal knowledge source we utilize. General knowledge on the other hand refers to structured information available from public knowledge repositories such as DBPedia, Freebase[14] and others.

4.3 Experimental evaluation

In PIMO, each entity can have a textual description. A current use-case for *seed* is its use as the text editing tool for these PIMO texts. Through the composition of natural language text description for entities, *seed* infers and suggests related entities for the user. Thus, saving the user the time and effort of manually searching for and annotating the entity at hand with related ones.

An evaluation of *seed* will be done in combination with the WP9 evaluation in the Fringe festival, where participants will use the PIMO and thus also *seed* to annotate their pictures and write down notes for images. The goal is to reduce manual annotation effort.

4.4 Software implementation

Following are implementation details about the various components of *seed*:

4.4.1 Editor component

The current implementation of *seed* is meant - in the first place - to run in the browser. Therefore, it is written in JavaScript and HTML on the UI-side and in Java on the server-side. It is also possible to embed *seed*'s editor in GUIs built using other languages. The only prerequisite is the availability of an HTML capable UI element to run the editor component.

seed's front-end is an extended version of the open-source HTML-based rich text editor CKEditor[15]. It allows for annotating and tracking arbitrary parts of the text. Interaction with the annotated parts is also possible.

4.4.2 NLP component

This component is implemented as a Java service that builds on the capabilities of two famous NLP toolkits; Stanford CoreNLP[16] and Apache OpenNLP[17]. It can perform:

- Named entity recognition:
  Named Entity Recognition (NER) is one of the most famous subtasks of information extraction. Its target is to locate and classify atomic elements in text into predefined categories such as people, organizations, and locations. *seed* relies on the Stanford named entity recognizer to perform multi-token named entity recognition. Given sample input text such as:

  ```
  John Doe lives in Germany
  ```

  the NLP component returns the text after recognizing entities in a form similar to the following:

```
<person>John Doe</person> lives in <location>Germany</location>
```

For entity recognition, we adopt a three-class model that distinguishes between entities of the types person, location, and organization. It is important to mention that those three classes serve as recommendations for the expected type of the suspect entity. It is possible through other components (e.g., LOD component) to identify entities of other types as well.

- Coreference resolution:
  Coreference is the case where multiple expressions in text refer to the same thing. In a sentence like "John made sure he returns early" the words "John" and "he" probably refer to the same person whose name is John. Coreference resolution refers to the task of identifying which words refer to which things in a text that has coreference. For a sample input text such as:

  ```
  John Doe lives in Germany. He works at Fruits United Ltd.
  ```

  seed would resolve the coreference and return the following output text:

  ```
  John Doe lives in Germany. John Doe works at Fruits United Ltd.
  ```

- Relation extraction:
  In addition, the NLP component of *seed* builds on ReVerb [18] to extract binary relations from the text being composed. It is currently limited to extracting binary relation phrases of the form:

  ```
  <Argument 1><Relation phrase><Argument 2>
  ```

  The extracted relations can be further utilized to enrich the user's knowledge or search for more information about the entities in the text.

The NLP component can be found at:
https://git.opendfki.de/seed/seednlpapi.

4.4.3 LOD component

The Linked Open Data (LOD) component of *seed* includes APIs for extracting information from LOD sources such as DBPedia and Freebase. This component can work independently or in combination with the NLP component to improve information extraction. Following are example tasks where this component is involved:

- Entity disambiguation
  Distinguishing between different entities that have similar or identical names.

- Relation extraction
  Searching for relations among entities.

- Context inference
  Finding contextual information about entities mentioned in the text. This helps in annotating the entity and in relating it to parts of the personal knowledge of the user.

4.4.4 PIMO component

This *seed* component includes APIs for interacting with the user's PIMO. The interaction scenarios include but are not limited to:

- Recognizing entities
- Extracting information about recognized entities
- Extracting relations between entities mentioned in the text

Figure 3 shows the editor component embedded in a Java GUI of the PIMO.
The following URL points to a PIMO exploration tool which employs *seed* as its text editor:

https://pimo.kl.dfki.de/forgetitsandbox/

A summary of the technical details of the software is shown in Table 3.

Table 3: Software technical details

Functional description	A semantic text composition tool
Input	Rich text
Output	Annotated rich text with and information about its content
Language/technologies	HTML, JavaScript, Java
Hardware Requirements	N/A
OS Requirements	N/A

4.5 Conclusions and future work

The current *seed* prototype serves as a proof of concept for semantic text composition. Planned improvements include:

- Reducing the need for user interaction required to review suggested entity annotations.
- Realizing more scenarios where the knowledge extracted from the text is used not only to enrich the composed text, but also to complement the user's personal knowledge (i.e., PIMO). These scenarios are addressed in deliverable D9.1 [2]. A scenario with high interest is to organize and annotate photo collections where users write descriptive texts, notes, or stories which have been told by their parents.

The *seed* component has already been included in the PIMO infrastructure as a text editor for texts written in the PIMO (see Figure 3), e.g., as meeting protocol or describing a thing. Future work will especially investigate the use cases for personal preservation.

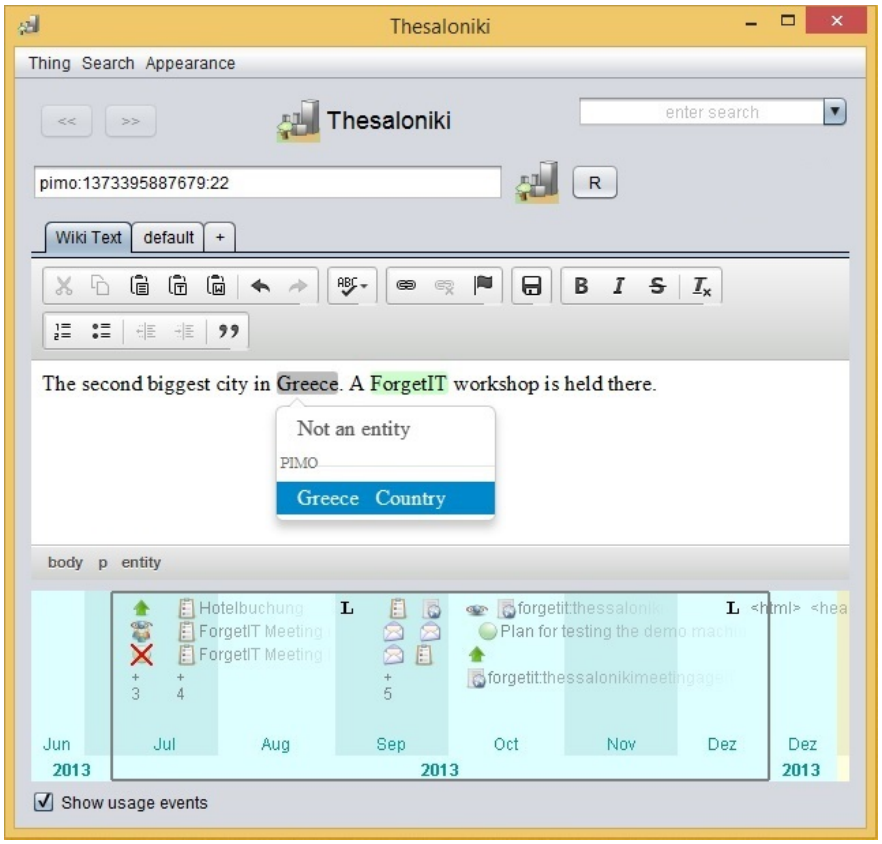

Figure 3: Snapshot of *seed* embedded in PIMO

5 Feature extraction and concept detection in image collections

5.1 Problem statement

High-level image representations are investigated in recent years in the scope of digital preservation and summarization. The huge amount of images captured in everyday life triggers the need of organizing image collections in a way that important images can be distinguished from less important ones and finally a new sub-collection can be constructed containing images that someone would like to preserve during the years.

The purpose of concept detection is to analyse the visual content of an image and automatically infer keywords (or tags) indicating the presence of semantic concepts. Initially, a vector representation of the images is created using feature extraction techniques. A wide variety of global and local features can be extracted from an image, but local features are preferred in the most recent literature approaches. For local feature extraction, first, either an interest point detection technique [19] (such as corner or edge detector) or a sampling approach [20], is applied in order to select points of interest in the image. Then, for each point of interest a descriptor is calculated. The well-known Scale-Invariant Feature Transform (SIFT) descriptor, presented in [21], is based on local image structure and its color variations - RGBSIFT and opponentSIFT, presented in [22], exploiting color information, are popular choices. To reduce computational cost, fast approximations of SIFT, such as Speeded Up Robust Features (SURF) [23] and DAISY [24] descriptors have also been introduced. The final representation of an image can be produced by encoding the local descriptors using the well-known Bag-of-Word (BoW) feature representation [25], or other recent approaches (e.g., VLAD [26], Fisher Vectors [27]). Finally, semantic concepts are detected using these image representations as input to machine learning techniques, such as trained Support Vector Machine (SVM) [28] classifiers, which produce a confidence score indicating the presence of the corresponding concepts in the image. We refer the reader to deliverable D4.1 [1] of WP4 for a detailed presentation of the state-of-the-art.

5.2 ForgetIT approach

A feature extraction and concept detection component has been developed in ForgetIT. As shown in Figure 4 the component input can be either an image or an image collection that contains N images. Two main processes are applied to the input images, as described in more detail in the sequel. The final output is a vector for each image whose elements are confidence scores (called Degree of Confidence (DoC)), indicating for each concept how much it relates to each input image.

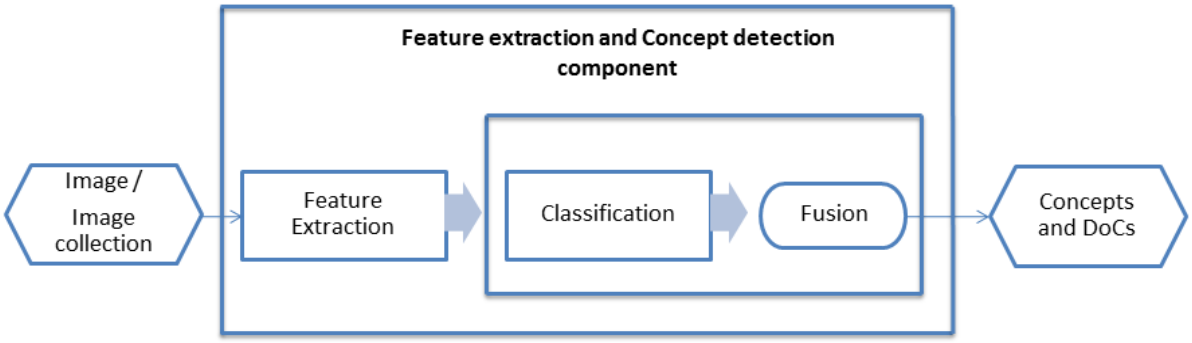

Figure 4: Feature extraction and concept detection component

5.2.1 Feature extraction

The feature extraction process takes as an input an image and extracts a Bag-of-Words vector representation. Four individual steps are performed sequentially in order to extract the output as shown in Figure 5. The most interesting part of this process is the descriptor, where in ForgetIT we introduce color variations of the SURF descriptor, aiming to exploit image color information and reduce the necessary processing time, compared to previous local color descriptors. The rest of the process is built based on well-known state-of-the-art approaches.

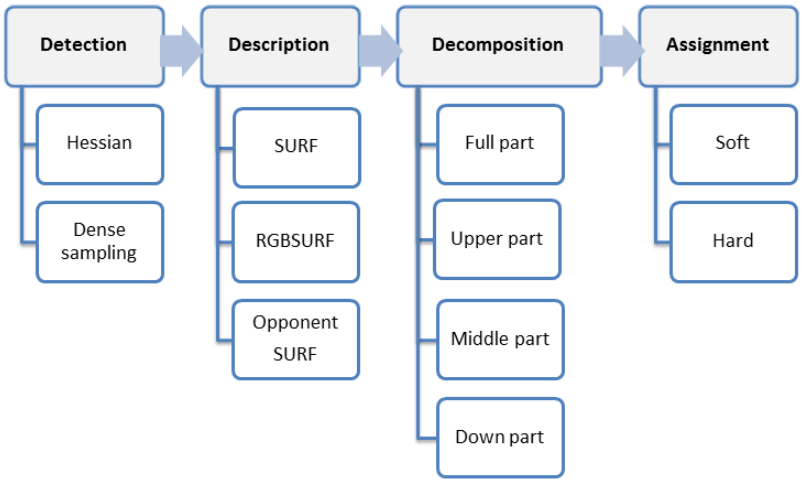

Figure 5: Feature extraction steps

Keypoint Detection: The extraction of low level features is performed on image points or regions of interest. Two types of detectors are used in our approach: i) the default keypoint detector that is used by the SURF descriptor, the edge based Hessian detector [23], where the number of the extracted interest points depends not only on the image characteristics but also on a threshold value (which the lower it is, the more keypoints are extracted) and ii) a dense sampling strategy where the number of keypoints is a constant

number for fixed image size.

Descriptor: In the descriptor part, a feature vector is extracted representing each interest point. We use an implementation of SURF for GPU from the OpenCV C++ library [29, 30], which extracts a 128-dimensional feature vector for each keypoint. In addition, in order to exploit color information we developed two color variations of SURF, the **RGBSURF** and the **opponentSURF**. In RGBSURF case, we first split the three RGB channels of the color image and then apply the original OpenCV implementation of SURF to each channel separately, while for the opponentSURF descriptor we first transform the RGB space to an opponent space and then proceed as above. Thus, three 128-dimensional feature vectors are extracted and concatenated to one 384-dimensional vector for each keypoint. The vectors, as extracted from the OpenCV implementation, contain double elements. We convert the vector elements to integer values by multiplying them with a constant factor and then round them to the closest integer (which has computational gains).

Pyramidal Decomposition: A 1×3 spatial pyramid decomposition scheme is used, i.e., the entire image is the pyramid cell at the first level *(Full part)*, and three horizontal image bars of equal size are the pyramid cells at the second level *(Middle-Upper-Down part)*. In this way, we divide the extracted vectors according to their location and hence we manage to maintain some spatial information of the image structure.

Assignment: The final step of the feature extraction method comprises two types of assignment, *hard* and *soft* assignment. Firstly, a pre-processing step is required, where a visual word vocabulary[1] is constructed off-line for each part of the images. Therefore, 4 Bag-of-Words (BoWs) with 1000 words each one is constructed. Then, hard assignment assigns each feature vector to the closest bin[2], while soft assignment assigns each vector to four closest bins.

The final outputs are 4×1000-dimensional vectors for each image. We concatenate them resulting in a 4000-dimensional BoW vector representation for each image.

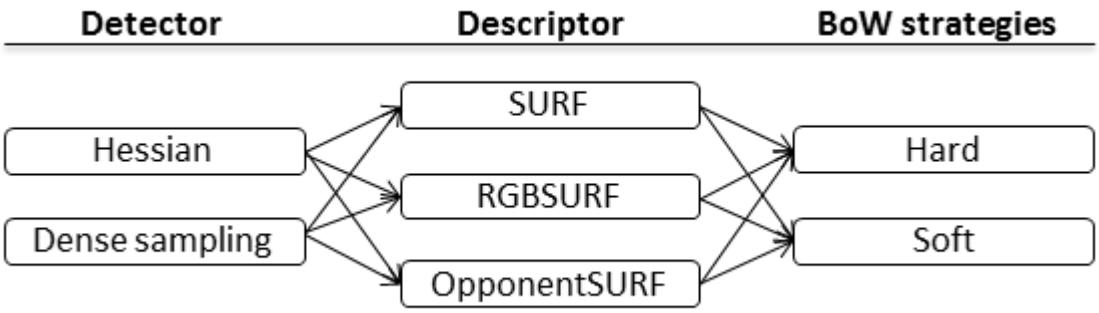

Figure 6: Configurations by combining detector, extractor and BoW strategies

[1] An image collection of 135.747 images were used for the BoWs construction. 250.000 feature vectors were extracted randomly and a kmeans clustering algorithm was applied giving 1000 visual words

[2] One of the 1000 visual words of the codebook

Finally we came up with 12 different combinations of 2 detectors (hessian, dense sampling), 3 descriptors (SURF, RGBSURF, opponentSURF) and 2 BoW strategies (hard-, soft-assignment), called configurations and presented in Figure 6.

5.2.2 Concept detection

The concept detection process consists of two individual steps (Figure 7) resulting in the extraction of the final Degree of Confidence (DoC) for each concept and each image of the collection.

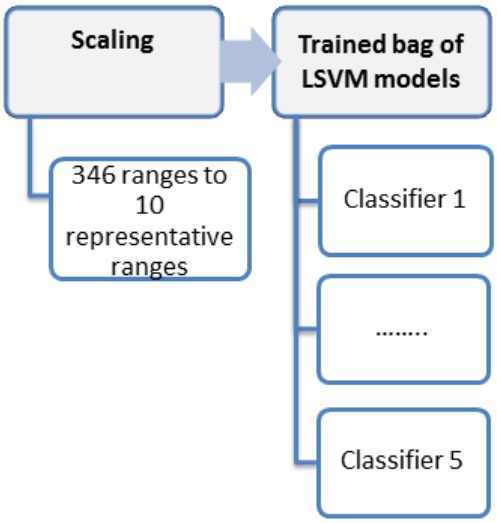

Figure 7: Concept detection steps

Training: Training of the SVM-based classifiers is performed with the use of the TRECVID 2012 Semantic Indexing Task [31] training data.

Scaling: The testing data are scaled using 10 representative ranges, which have been calculated at the training phase, selected among all ranges that were estimated during training for the entire set of concepts. This represents a good compromise between SVM accuracy (which benefits from task-specific scaling of the data) and computational complexity (scaling being one of the most computationally demanding phases of concept detection).

Prediction: The scaled testing data are passed to the LSVMs. The number of employed SVMs per concept ranges from 5 to 60 depending on the number of configurations that we use in each experiment (1 to 12 configurations). Each LSVM returns one prediction score in the range [0,1] expressing the DoC (higher values are better). The output for one configuration and for each image is a model vector of N elements (where $N = 46$ is the number of concepts in our initial experiments).

Late fusion: A late fusion strategy is applied to combine the DoCs of the different config-

urations for the same concept. This fusion is realized by calculating either the arithmetic or the harmonic mean of the DoCs.

5.3 Experimental evaluation and comparison

For evaluating the proposed approach, we compared the proposed color SURF variants with the corresponding SIFT descriptors. The results are presented in Tables 4 and 5. These results show that the proposed color SURF variants compare favourably to the corresponding SIFT descriptors, contributing to faster and more accurate concept detection.

The experimental set up employs the entire video dataset and the concept list that were used in the 2012 TRECVID SIN task. The video dataset comprises 8263 videos of almost 200 hours of total duration which are pre-segmented into 145634 shots. As in TRECVID, 46 semantic concepts were evaluated out of a total of 346 concepts due to the unavailability of ground truth annotations for all 346 concepts. We evaluated our approach in terms of concept detection accuracy and execution time.

All experiments have been executed on the same computing environment, namely Windows 7 OS 64-bit machine, Intel Core (TM) i7-3770K CPU@ 3.50GHz processor, 16 GB RAM, CUDA [32] processing for the GPU acceleration.

Table 4: Execution times (in seconds) between SURF and SIFT implementation

Configuration		Execution times (seconds)
Detector	**Descriptor**	
hessian	SURF	630
harris-laplace	SIFT	1271
hessian	RGBSURF	1543
harris-laplace	RGBSIFT	3009
hessian	opponentSURF	1513
harris-laplace	opponentSIFT	2913.5
dense sampling	SURF	1898
dense sampling	SIFT	2112
dense sampling	RGBSURF	5083.5
dense sampling	RGBSIFT	5701.5
dense sampling	opponentSURF	5365
dense sampling	opponentSIFT	5915

Table 5: Extended Inferred Average Precision (xinfAP) [33] of individual and fused configurations. Higher values of xinAP are better.

Number of configurations	Interest point detector	descriptor	BoW strategy	xinfAP (%)
1	dense sampling	SURF	soft	6,97
1	dense sampling	SIFT	soft	6,08
1	dense sampling	RGBSURF	soft	7.86
1	dense sampling	RGBSIFT	soft	7,02
1	dense sampling	opponentSURF	soft	7,33
1	dense sampling	opponentSIFT	soft	7,12
fusion of 3	dense sampling	SURF RGBSURF opponentSURF	soft	12,87
fusion of 3	dense sampling	SIFT RGBSIFT opponentSIFT	soft	10,81
fusion of 6	dense sampling	SURF RGBSURF opponentSURF	hard soft	13
fusion of 6	dense sampling	SIFT RGBSIFT opponentSIFT	hard soft	10,57
fusion of 6	hessian	SURF RGBSURF opponentSURF	hard soft	9,1
fusion of 6	harris-laplace	SIFT RGBSIFT opponentSIFT	hard soft	9,1

5.4 Software implementation

We developed a Web Service on a server located in CERTH which executes the feature extraction and concept detection method. In this service the user can give a URL address of a compressed file containing an image collection. This compressed file is downloaded locally to our server where the images are processed. There are no limitations in terms of image size, image format (we tried to include all well-known image formats, such as, .jpeg, .png, .bmp, .tif, .gif.) and archive format (the supported formats are .zip, .rar, .tar, .gzip). Furthermore, our software can deal with cases where the provided compressed archive contains, besides the images, other document types, such as text files, xml files, etc. In this case, only the image files are processed and the rest of the files are inevitably ignored by this component. The output of our Service is an XML file, which is returned to the user's browser containing the concepts and the corresponding confidence scores

of each image. The user can save this XML file and use it for further processing. A summary of the technical details of the feature extraction and concept detection web service is shown in Table 6.

Table 6: Summary of the technical details of the implemented service

Functional description	Feature extraction and concept detection in image collections
Input	An image collection contained into a compressed folder
Output	A text and XML file containing the DoC
Language/technologies	JAVA Web Service
Hardware Requirements	NVIDIA graphic card-CUDA drivers
OS Requirements	Windows

The URL query through which the service is called is: `http://multimedia.iti.gr:8080/ForgetITImageConceptDetectionService/ImageConceptDetection?pathType=''&pathName=''&analysisTech=''&modelFlag=''&confFlag=''`

where, **pathType** is the type of the path through which the image collection will be provided, (currently only the *typeUrl* type is available), **pathName** is the path of the image collection location, **analysisTech** is a flag indicating the type of the input items (currently these can be only images; at a later stage, processing video will also be supported), **modelFlag** is a boolean flag (true,false) indicating which prediction approach will be used, **confFlag** is an integer flag (1,2,3) indicating which configuration set will be used.

In Table 7, the values that can be assigned to the modelFlag and confFlag are presented, as well as a description of the approach that is followed by choosing each of them. In our first demo version of feature extraction and concept detection method only modelFlag equal to true and confFlag equal either 1,2 or 3 values are available.

Table 7: modelFlag boolean values (true,false) and confFlag integer values (1,2,3)

modelFlag	
true	*false*
BoMs per concept and 10 representative ranges for all concepts	1 classifier per concept and 1 range for all concepts

conflag	**detector**	**descriptor**	**BoW strategy**
1	dense sampling	RGBSURF	soft
2	dense sampling	SURF RGBSURF opponentSURF	soft
3	dense sampling hessian	SURF RGBSURF opponentSURF	hard soft

An example of the URL query is: `http://multimedia.iti.gr:8080/`

ForgetITImageConceptDetectionService/ImageConceptDetection?pathType=typeUrl&pathName=http://vision.okstate.edu/csiq/src_imgs.zip&analysisTech=images&modelFlag=true&confFlag=1

where an image collection, called "src_imgs.zip" and located in "http://vision.okstate.edu/csiq/", is passed as an input to our concept detection approach, one configuration (RGBSURF-dense-soft) will be executed and the prediction will be performed using 5 classifiers for each concept. A representative example of an input image and part of the XML result is given in Figure 8.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<Concept_detection>
    <Concepts_list>
        <concept id="C1">Airplane</concept>
        <concept id="C2">Airplane_Flying</concept>
                    ..................
        <concept id="C44">Oceans</concept>
        <concept id="C45">Skier</concept>
        <concept id="C46">Soldiers</concept>
    </Concepts_list>
        <Concepts_order>C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19
                C20 C21 C22 C23 C24 C25 C26 C27 C28 C29 C30 C31 C32 C33 C34 C35 C36 C37 C38 C39 C40
                C41 C42 C43 C44 C45 C46</Concepts_order>
     <Image_Concepts_List>
         <Image user_id="user_469\woman.png">
           <Collection_folderName>src_imgs.zip</Collection_folderName>
           <image_id>1</image_id>
           <confidence_scores>0.01320 0.00364 0.00894 0.37634 0.02441 0.79165 0.0074 0.00249 0.0017 0.20267
        0.45219 0.00117 0.10272 0.01059 0.00605 0.04524 0.31182 0.59674 0.21125 0.20439 0.0011 5.677e-005 0.00025
        0.0196 0.14952 0.06558 0.06643 0.02036 0.27392 0.00995 0.58364 0.4087 0.11387 0.22629 0.19602 0.92033
        0.12692 0.00037 0.01069 0.36343 0.00347 0.00109 0.00046 0.03451 0.00023 0.01944</confidence_scores>
         </Image>
                    ..................
      </Image_Concepts_List>
</Concept_detection>
```

Figure 8: Input image and part of the XML output file

5.5 Conclusions and future work

Our work on feature extraction and concept detection currently focuses on image collections. In our future work we will extend this to video collections as well. Moreover, we will improve our approach experimenting with different descriptor representations (e.g., VLAD, Fisher Vectors).

6 Image quality assessment

6.1 Problem statement

Image quality is a measure of the perceived degradation of the visual content of an image. There are several distortions that an image may undergo, either during its capturing or during subsequent post-processing, such as image blurring, contrast decrease/increase, noise addition etc. Image quality assessment (IQA) techniques aim to quantify the amount of image degradation and classify the images according to their visual quality. Detection and possibly removal of the distorted images from an image collection can control and enhance the multimedia preservation process. Therefore, the efficient quantification of visual image quality is an appealing and challenging field of research. Various techniques which exploit the image transformations (i.e., Discrete Cosine Transform (DCT), wavelet, etc.), image histogram analysis, Natural Scene Statistics (NSS) and supervised learning models can be found in the relevant literature. For detailed information on the state-of-the-art IQA techniques, please refer to the deliverable D4.1 of WP4 [1].

6.2 ForgetIT approach

ForgetIT developed an IQA approach for images, presented in Figure 9. The examined image quality measures include blur, contrast, darkness and noise and they constitute the four main processes of the component. As shown in Figure 9, this method takes as an input an image collection, it processes each image separately for each quality measure and gives as an output an image quality score.

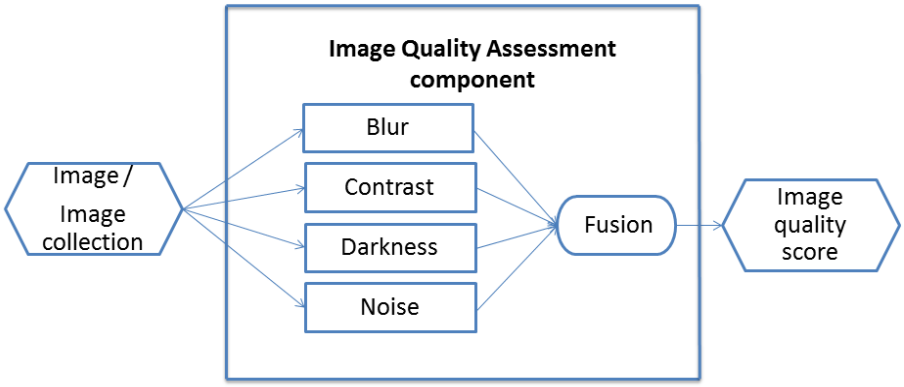

Figure 9: Image quality assessment component

Blur: Image quality degradation due to the presence of blur is a common problem, often caused by the slow shutter speed, out-of-focus of the lens, or the relative motion between the camera, a moving object and the background. In order to detect the blurred images, we have developed an image blur assessment approach which exploits the information

derived from the frequency spectrum of the image. The original image is partitioned into 9 equal blocks according to the rule of thirds. Subsequently, the Fourier transform of the entire image and each of the 9 image patches is computed in order to extract the appropriate information about their frequency distribution. We achieve the quantification of high frequencies distribution by subdividing the frequency amplitude according to the following ranges: [1, 100], [100, 150], [150, 200], [200, 300] and [300, max] and calculating this frequency histogram for each of the ten images (the initial image and the 9 image patches). Finally, all the aforementioned histogram bins are concatenated in a vector which serves as the input to an SVM classifier which provides a confidence value indicating the probability of an image being blurred. In Figure 10 the overal scheme of the blur detection method is presented.

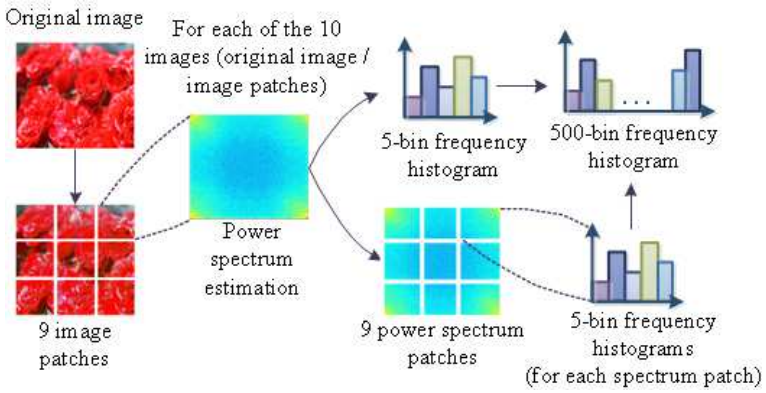

Figure 10: Overall scheme of the blur detection method

Contrast: Under certain shooting conditions such as daylight, mist or fog, captured images may have low contrast. Low contrast images may be characterized as flat or impure and they are not appealing to the viewers. The most common contrast measures appearing in the relevant literature are the Root Mean Square (RMS) contrast, the Michelson contrast and the Weber contrast [34]. In our approach, we used the first two measures. As mentioned in deliverable D4.1 [1], the Michelson contrast measure presents weaknesses due to its dependence on the luminance value extrema. To prevent this weakness we developed an improved approach, which estimates the average max and min luminance values in the horizontal and vertical direction of the image and computes the final contrast score as the average of Michelson contrast in these two directions,

$$C_{Mich,d} = \frac{I_{max,d} - I_{min,d}}{I_{max,d} + I_{min,d}}, d = vertical, horizontal.$$

Besides the RMS and the improved Michelson contrast, we have estimated the distribution of RGB channel's values into the range of [0, 50] and [150, max], which determines the presence of contrast. Additionally, the saturation in HSV color format which represents the chromatic purity, is estimated:

$$Sat = \frac{1}{XY} \sum_{x=1}^{X} \sum_{y=1}^{Y} S(x, y),$$

where X, Y are the height and the width of the image respectively. Finally, all the aforementioned features are concatenated in a vector forming the input to an SVM regression model which provides a confidence value indicating the probability that an image has low contrast.

Darkness: Browsing an album of images captured for example during a concert, we can observe that there are many dark photos which not clearly depict the subject of interest. During the night and under low light conditions the image capturing may lead to black images or to images with very low brightness which do not present meaningful content. The ForgetIT approach exploits the histogram information of the images in order to quantify the amount of darkness. In particular, we examined the information derived from the YCbCr and HSV color formats, employing the Y and the V channels which represent the luminance and the color intensity of an image respectively. Each image is partitioned into 9 equal blocks according to the rule of thirds and the histogram of the luminance values into the range of [0, 50], where the low luminance pixels are allocated, is estimated for both the entire image and each of the 9 blocks. Furthermore, the mean pixel intensity of the entire image and the middle image patch is estimated:

$$V_{mean} = \frac{1}{XY} \sum_{x=1}^{X} \sum_{y=1}^{Y} I_V(x, y).$$

Finally, the aforementioned features are concatenated in a vector forming the input to an SVM regression model which provides a confidence value indicating the probability that an image is dark.

Noise: Images during their capture or during several compression processes are prone to noise degradation. Noise can significantly degrade the perceived image quality as it appears in the form of speckled pixels of color on the image's surface. In order to detect and quantify the amount of image noise we use the Blind Image Quality Index's (BIQI) technique [35] whose code is publicly available. This method employs NSS in the frequency domain and applies a supervised learning approach in order to quantify the image quality. Specifically, first a wavelet transformation is performed on the examined images over three scales and three orientations. Subsequently, based on NSS, the extracted information is modelled using a Generalized Gaussian Distribution (GGD). The GGD parameters, which are estimated for each of the three scales and three orientations, form the input of an SVM regression model which produces a confidence value for the image quality. BIQI provides a specific model for noise quantification which is finally used in the ForgetIT IQA approach.

Fusion: After the calculation of each individual image quality measure, a final quality score is computed using the Minkowski sum over the 4 quality measures. The fusion of blur, contrast, darkness and noise quality measures is performed by the following equation:

$$ImageQualityScore = (\sum_{i=1}^{4} b(i) * ImageQualityMeasure(i)^p)^{\frac{1}{p}}$$

The parameters *b* and *p* were chosen carefully in order to optimize the measure's per-

formance in terms of correlation with human Mean Opinion Score (MOS). The output of the implemented visual quality assessment component consists of the four separate confidence scores indicating the probability that each image distortion is present and the result of their fusion. The quality scores typically have a value between 0 and 1 (0 represents the best quality, 1 the worst).

6.3 Experimental evaluation and comparison

For the training and the evaluation of the aforementioned quality measures, we created a large image dataset consisting of more than 3000 digital photographs. Specifically for the image blur assessment we created a training set consisting of 1000 images (blurred and undistorted) and two evaluation sets. The "Natural blur" testset consists of 1000 natural images captured by various camera models and the "Artificial blur" testset consists of 480 images which are artificially distorted. In Table 8, we present the experimental results of the blur detection approach, as the blur is the most common image quality distortion type, and we compare its results with the blur assessment techniques BIQI and Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) [36]. Blur detection approach, as shown by the experimental results, presents noticeable performance for both artificially and naturally-blurred images.

Table 8: Experimental results of the blur detection method

	"Natural blur" testset			**"Artificial blur" testset**		
	ForgetIT approach	**BIQI**	**BRISQUE**	**ForgetIT approach**	**BIQI**	**BRISQUE**
Accuracy	**0.8720**	0.7470	0.6290	**0.9917**	0.9417	0.9729
Precision	**0.8394**	0.6763	0.5333	**1**	0.9797	0.9760
Recall	**0.8394**	0.7372	0.7786	0.9911	0.9644	**0.9956**
F-score	**0.8394**	0.7055	0.63330	**0.9955**	0.9720	0.9857

6.4 Software implementation

We developed a Web Service on a server located in CERTH which executes the image quality assessment method. The user can give a URL address of a compressed file containing an image collection. This compressed file is downloaded locally to our server where the images are processed. Similarly to our concept detection component (Section 5.4) there are no limitations in terms of image size and image format, and media items of non-image types (e.g., text files, XML files, etc) are ignored. The output of our Service is an XML file. A summary of the technical details of the IQA web service is shown in Table 9.

The URL query which call the service is:

`http://multimedia.iti.gr:8080/ForgetITImageQualityAssessmentService/ImageConceptDetection?pathType=""&pathName=""`

Table 9: Summary of the technical details of the implemented service

Functional description	Image quality assessment of image collections
Input	An image collection contained into a compressed folder
Output	A text and XML file containing the quality DoC
Language/technologies	JAVA Web Service
Hardware Requirements	N/A
OS Requirements	Windows

where, **pathType** is the type of the path through which the image collection will be provided, (currentrly only the *typeUrl* type is available) and **pathName** is the path of the image collection location.

An example of the URL query is: `http://multimedia.iti.gr:8080/ForgetITImageQualityAssessmentService/ImageConceptDetection?pathType=typeUrl&pathName=http://multimedia.iti.gr:8080/CERTH_`

```
<QualityMeasure_detection
    <QualityMeasures_list>
        <QualityMeasure id="C1">Blur</QualityMeasure>
        <QualityMeasure id="C2">Contrast</QualityMeasure>
        <QualityMeasure id="C3">Darkness</QualityMeasure>
        <QualityMeasure id="C4">Noise</QualityMeasure>
    </QualityMeasures_list>
    <QualityMeasures_order>C1 C2 C3 C4</QualityMeasures_order>
    <Image_QualityMeasures_List>
        <Image user_id="user_3\DSC07040.jpg">
            <Collection_folderName>test</Collection_folderName>
            <image_id>1</image_id>
            <confidence_scores>0.96757 0.13545 0.33641 0.277</confidence_scores>
            <imagequality_score>0.90278</imagequality_score>
        </Image>
        <Image user_id="user_3\DSCN0470.JPG">
            <Collection_folderName>test</Collection_folderName>
            <image_id>2</image_id>
            <confidence_scores>0.98283 0.51943 0.3489 0.30859</confidence_scores>
            <imagequality_score>0.91711</imagequality_score>
        </Image>
    </Image_QualityMeasures_List>
</QualityMeasure_detection>
```

Figure 11: Input image and part of the XML output file

BIN/ForgetIT_Quality_Assessment_Service/test.zip

where an image collection called “test.zip” is given as input to our image quality assessment Web Service. Running this example, the results of the IQA process will be returned to the browser in the form of an XML file. A representative example of an input image and part of the XML result is given in Figure 11.

6.5 Conclusions and future work

ForgetIT IQA method focuses entirely on an images. In future work we are aiming to extend it in order to be able to evaluate video quality as well. Moreover, the existing measures could be further improved in terms of prediction accuracy, and our goal is to incorporate them into a method which will eventually also assess the image aesthetics.

7 Face detection for clustering

7.1 Problem statement

Face detection is the process which takes as an input an image and returns the location of the face/s which is/are contained in it. Usually, the location is provided in the form of coordinates of the face bounding box. At first glance, the only information that face detection can provide is the number of faces and their relative proximity to the camera (obtained from the bounding box sizes and assuming that the actual face sizes are more or less equal). However, by employing further image analysis algorithms, we can calculate the similarity or distance between facial regions and then cluster them according to this measure. The output of the aforementioned procedure is the identification of the existence of the same person in several images. By counting the number of appearances of the detected faces (number of elements of each cluster) we can locate the dominant faces which are of higher importance in our image dataset. In order to increase the accuracy of the clustering method we have to provide as much accurate as possible input faces extracted from the face detector. Thus, initially we focus our effort on creating an efficient and accurate face detector.

7.2 ForgetIT approach

As presented in Section 5.2 of deliverable D4.1 [1] Viola and Jones detection [37] approach and its extensions are among the most successful methods presented in the literature. They can be used to detect not only faces, but also other objects (following appropriate training). As a result, other facial features such as nose, mouth and eyes can be detected as well using the same algorithmic approach. Furthermore, more than one face detectors can be employed in order to increase the number of successfully detected faces.

Our approach studies the use of more than one face detectors as well as the verification of the detected faces through facial characteristics detection.

Decreasing face detector errors

The errors that can arise from face detection techniques are false alarm and false rejection errors. The former refers to an erroneous identification of a region which does not contain any face, as a facial region. The latter refers to the failure to find existing facial regions.

Since the output of face detection will be used in face clustering, it is very important to decrease as much as possible the face detection errors. It is obvious that if the quality of the face detector is poor, then face clustering will not be able to provide high quality results.

In order to decrease the missed detection error, we used a number of face detectors to capture as many faces as possible. We examined the face detectors separately and we have also employed their union in order to increase the number of detected faces.

Detecting facial characteristics By adopting the union of face detectors we manage to decrease the missed detection error. However, the false alarm is increased significantly since the detected faces set includes a very high number of regions that do not contain faces. In order to cope with this issue, we consider the detected faces as potential facial regions and, in every region, we try to detect other facial characteristics. If other facial characteristics are detected then the detected region is classified as face. The facial characteristics which we search for in the potential facial regions are eyes, nose and mouth. In our experiments we have tested the 9 following rules according to which we accept or reject a potential facial region as a face.

The detected region is accepted as facial:

1. always (regardless the existence or not of other facial features)
2. if eyes are detected
3. if mouth is detected
4. if nose is detected
5. if eyes and mouth are detected
6. if eyes and nose are detected
7. if mouth and nose are detected
8. if eyes and mouth and nose are detected
9. if eyes or (mouth and nose) are detected

Facial characteristics detection fine tuning In many cases there are false detections of facial characteristics in no facial areas. In order to eliminate these false detections we check if their spatial location is in agreement with the physical location of them in a face. For example, all the features should be horizontally centered in the facial region, eyes can not be located too low, mouth can not be located too high etc. Thus, as a first step we reject the facial characteristics that do not comply with the physical location. Then, we accept or reject the facial area based on the type of the accepted facial characteristics (eyes, mouth or nose) and the selected acceptance rule (one of the 9 rules listed above).

7.3 Experimental evaluation and comparison

Database

We have tested our method in Gallagher Collection Person Dataset [38]. It is a set of 589 images which are typical digital image snapshots, captured in real life, at real events,

of real people, with real expressions. In the provided ground truth file, 931 faces with 32 identities have been labeled using as a labeling criterion the fact that the eyes must be visible, and the face must be ”mostly” frontal (i.e. both ears should be visible, except for occlusion). Labelling is given in the form of a text file that contains the coordinates of the eye pairs locations of each image and the index of each of the 32 identities. It contains 931 rows, one per detected face and each of them has the form $\{$ImageName, LeftEyeXCoord, LeftEyeYCoord, RightEyeXCoord, RightEyeYCoord, IdentityIndex$\}$.

Experimental results

We run experiments with 6 face detectors and the 9 rules discussed in Section 7.2. For each detector and for each detected facial region we applied the facial characteristics detectors. After rejecting those facial characteristics for which their spatial location is not in agreement with the physical location in the face, the detected faces are accepted according to the valid facial characteristics and acceptance rule. For example, a facial region that contains only eyes, would be accepted in the cases that the acceptance rules are the 1st, 2nd, 5th, 6th, 8th of 9th and rejected otherwise.

For face detection we have used the detectors proposed in [39] and [40], while for eyes the nose and the mouth the method introduced in [41]. The notations used in the table correspond to the classifiers as follows: hfa2: haarcascade_frontalface_alt2, hfa: haarcascade_frontalface_alt, hfat: haarcascade_frontalface_alt_tree, hfd: haarcascade_frontalface_default and FF_LB: FrontalFaceLBP. A ”merged” detector has been also employed which is actually the union of the detections of the aforementioned detectors.

For each pair of face detector-acceptance rule, we have calculated the precision and the recall. Recall R indicates the percentage of faces that have been detected while precision P the percentage of detected objects that are faces. Obviously, the higher the recall and the precision are the more reliable and accurate the face detection method is. If we target to increase recall, then the number of the real faces that are detected is increased but, on the other hand, there is also an increased number of false detections. If we aim to obtain high precision values, then less real faces are detected but the number of false detection is decreased as well. Since our objective is to use the face detection output as an input of face clustering our goal is to include as less as possible false detections namely to increase the precision. Attempting to combine the two measures, precision and recall into a single metric, we calculated the F-measure using the

$$F = \frac{2R \cdot P}{R + P}.$$

In table 10 the F-measure values for all combinations of detectors/rules are presented.

F-measure has high values (above 80%) for rules 1,2,4 and 9. However, in some of these cases the corresponding precision value is too low. Thus, due to the high significance of precision values, in Table 10 we marked (used underlined figures) the F-measure values for which the corresponding precision value is above 94%. From these results we observed that the face detector is more reliable (in terms of F-measure) for rules 2,4 and

Table 10: F-measure values for all combinations of detectors/merging rules

F measure (%)									
	rule								
Detector	1	2	3	4	5	6	7	8	9
hfa2	87.69	83.54	67.37	87.30	58.34	79.42	61.78	53.24	88.50
FF_LBP	81.52	82.31	64.94	85.22	56.87	78.12	60.11	51.93	87.07
hfa	88.96	82.65	66.76	87.05	58.23	79.41	62.67	55.10	**87.23**
hfat	85.32	78.26	63.45	81.36	55.82	73.67	57.97	51.40	82.51
hfd	68.91	82.59	65.77	82.81	58.72	76.79	60.96	53.19	86.91
merged	64.31	81.54	68.06	82.90	62.10	76.00	65.50	57.66	86.38

9. The results above also show that for rule 9 the highest values of recall are achieved (above 80%). From the results above we selected the combination hfa-rule 9.

7.4 Software implementation

We developed a Matlab® [42] function which performs image face detection. For each image it can detect faces using more than one classifiers and give as an output the face bounding boxes. Also, it calculates the "merged" detector which takes into account all the tested classifiers' output. Then, for each detected face, facial characteristics (mouth, eyes and nose) are detected using the user selected classifiers and the results that do not comply with their physical location are eliminated. Finally, each face is accepted or rejected by applying one of the nine rules listed in Section 7.2. The results are saved in a variable and in a text file. The prototype component described in this section can be downloaded from `http://www.forgetit-project.eu/en/downloads/workpackage-4/`. Please contact the project for access to this protected section of the website.

The `face_detection_ForgetIT` function is called as follows:

```
output=face_detection_ForgetIT( directory_path,facemodels,facial_elem_-
models, rule ,filename);
```

where

- `directory_path` is the path that the images are contained,
- `facemodels` is a $n \times 2$ cell array. Its first column contains the face detector model names (matlab model strings or xml file names), while the second one user defined names (e.g. face_detector_1 etc)
- `facial_elem_models` is a 1×3 cell array containing the facial elements detector where the first elements corresponds to the eye, the second to the nose and the third to the mouth detector

- `rule` takes a value from 1 to 9 according to the acceptance rule
- `filename` is the name of the file that the result will be stored
- `output` is a struct that contains all the results

Figure 12: Face detection

Execution example:

```
output=face_detection_ForgetIT(impath,...
{{'FrontalFaceCART','FrontalFaceCART'},
{'FrontalFaceLBP','FrontalFaceLBP'}},...
 {'EyePairBig','Nose','Mouth'},1,'results.txt');
```

In this example, the images are located in a directory given by the parameter `impath` while the user has selected 2 face classifiers `FrontalFaceCART` and `FrontalFaceLBP` (using them also as user defined names) and for eyes, nose and mouth the `EyePairBig, Nose,Mouth` respectively. Note that instead of Matlab classification models either for face or for facial elements, the user can employ XML files containing custom classification models. The available Matlab classification models are listed in the help section of the `face_detection_ForgetIT` function while a list of available XML models are also included in the software package. Finally, the next argument (`1`) defines the acceptance rule (values 1-9 - see rules list in page 37) and the final argument the name of the output file. The output file has the following form:

```
1 1 1 457 322 693 693
1 2 1 459 312 694 694
1 3 1 457 312 696 703
2 1 2 1000 270 256 256 883 1109 283 283
2 2 3 1547 182 202 202 1008 277 251 251 889 1112 280 280
2 3 3 1000 270 259 258 883 1109 286 283 1547 182 202 202
3 1 1 445 666 140 140
3 2 4 1063 1042 50 50 463 1417 55 55 717 1010 77 77 1003 1543 110 110
3 3 5 445 666 140 140 1063 1042 50 50 463 1417 55 55 717 1010 77 77 1003 1543 110 110
...
```

where, the first column is the image index, the second the face detection classifier index, the third the number of detected faces and then, quartets of coordinates follow for each detected face. Each quartet defines the extracted bounding box where the first pair of numbers are the x, y coordinates of the top left bounding box corner while the second pair is the width and height of the bounding box. It should be noted that the number of classifiers is always one more than the user's input since the 'merged' classifier is also added. A summary of the technical details of the software is shown in Table 11. Also, a face detection output is illustrated in Figure 12.

Table 11: Summary of the technical details of the software

Functional description	Face detection of Image collection
Input	A directory that contains image files
Output	A txt file with face detections
Language/technologies	MATLAB R13a, Computer Vision Toolbox
Hardware Requirements	N/A
OS Requirements	Windows

7.5 Conclusions and future work

In the second year of the project we aim to increase the face detector accuracy and test it in more databases as well as to create a clustering method being able to group each person's faces within an image collection.

8 Image clustering for summarization

8.1 Problem statement

Clustering partitions a collection of images into groups called clusters, such that similar images are assigned into the same group. As a result, large datasets can be handled in such a way that images can be organized, and subsequently summarized by selecting images that belong to different clusters for constructing a summary.

8.2 ForgetIT approach

We developed a component which takes as input an image collection, applies a clustering algorithm and extracts the output, which is a set of clusters. In Figure 13 the image clustering component consisting of the input, the output and two intermediate processes, is shown. The first process is actually a pre-processing step which extracts a vector representation for each image of the image collection that is to be clustered. This step can be considered as the input of the component instead of an intermediate process depending on the type of the vector representation that will be used, namely, as we will discuss in the following of the section, model vectors extracted by concept detection method may be used by the clustering method. In that case, this step is actually the output of another component developed for the ForgetIT project and not a process inside the current component. The second process is the clustering algorithm, which constructs the N clusters and assigns each image to the cluster that is visually, or conceptually closest. Finally the output is the N clusters and the images assigned in each of them. In addition, the most representative image of each cluster (the one that is closest to the cluster center) is also extracted. Below we will discuss the clustering algorithms and the input data types that we have studied and evaluated.

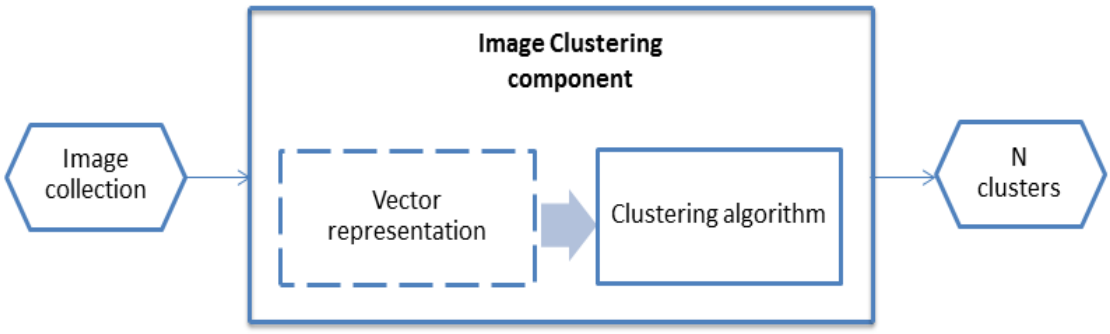

Figure 13: Image clustering component

As far the clustering algorithms are concerned, many approaches have been introduced in the literature, see deliverable D4.1 [1] for more details. In our experiments so far we

studied and evaluated 6 of them as follows:

- kmeans [43]
- Hierarchical clustering using complete linkage (hier-comp) [44]
- Hierarchical clustering using single linkage (hier-single) [45]
- Partitioning Around Medoids (PAM) [46]
- Affinity Propagation (AP) [47]
- Farthest First Traversal Algorithm as a 2-approximation for the k-center clustering (farthest first) [48]

The most interesting part of our approach focuses on the type of input data that is passed to the clustering process. The first type of input data that we studied was the commonly used HSV Histogram of 144 elements. Subsequently more complex data types, namely the BoW vectors and the model vectors (i.e., the outcome of visual concept detection) were evaluated. The former were extracted using dense sampling strategy for interest point detection, SIFT descriptor for feature vector extraction and soft assignment as BoW strategy. The final BoWs vector representation is a 4000-element vector per image/video keyframe. In the case of model vectors, their extraction follows the classification task described in Section 5.2.2. As for the training phase, we trained our framework on the TRECVID 2013 Semantic Indexing (SIN) ground-truth annotated development set. The 346 concepts that we used have been provided by TRECVID SIN task. Thus, each image/video keyframe is represented by a 346 element vector of confidence scores.

A well-known limitation of most clustering algorithms is the definition of the number of clusters that an image collection will be divided each time. In our first approach we are not dealing with this limitation; the number of clusters in our experiments was user-defined.

8.3 Experimental evaluation and comparison

We applied the clustering methods in 4 image collections whose size varies between 107 and 254 images. Additional experiments have been performed on 5 collections of video keyframes whose number of videos varies from 11 to 27. A shot segmentation algorithm has been applied to each video resulting in video frames and then we selected the most representatives from them as the keyframes. In the end, joining the 4 image and 5 video keyframes collections, we came up with totally 9 image collections. All images were processed separately to extract the three input data types.

The evaluation of the clustering results is performed by calculating the Normalized Mutual Information (NMI) value [49] between the automatic clustering and the manually created cluster ground truth. The NMI value ranges from 0 to 1 and higher values are better.

Two ground truth clustering have been created for each of the 9 collections, one for the same number of clusters (equal to 10) for all collections, and one for a collection-specific, user-defined number of clusters.

We evaluated all possible combinations of the 6 clustering algorithms and 3 input data types mentioned above. From these results we concluded that the optimum combination is the kmeans algorithm with model vectors, giving the best result in 4 out of 8 experiments in image collections and 7 out of 10 experiments in video keyframe collections. Due to lack of space, Table 12 presents the results of one representative experiment where the number of clusters is set to 7 and the NMI is calculated for each input-clustering algorithm combination. For almost all clustering algorithms, the use of model vectors results in the best performance among the three input data types-features. Furthermore, among the tested clustering algorithms, kmeans is one of the best for all input data features. Combined with model vectors, kmeans achieves the highest NMI value in this experiment.

Table 12: Clustering results

algorithm	Input data feature	NMI
kmeans	hsv	0,2653
kmeans	BoW	0,2361
kmeans	model vectors	**0,5979**
hier-comp	hsv	0,1778
hier-comp	BoW	0,1912
hier-comp	model vectors	0,5148
hier-single	hsv	0,1317
hier-single	BoW	0,1885
hier-single	model vectors	0,1073
PAM	hsv	0,2957
PAM	BoW	0,197
PAM	model vectors	0,4959
AP	hsv	0,2928
AP	BoW	0,2403
AP	model vectors	0,5499
farthest first	hsv	0,1669
farthest first	BoW	0,2164
farthest first	model vectors	0,464

8.4 Software implementation

We developed a Web Service on a server located in CERTH which executes the image clustering method. The user provides as input not the image collection itself but a text file containing a model vector representation for each image. Thus, in order the user to exe-

cute the image clustering method he/she first needs to call the ForgetIT concept detection Service (see subsection 5.4 Software Implementation). By calling the concept detection Web Service the URL address of the required text file is returned to the user's browser. Moreover, when the concept detection service is executed, the names of the processed images are changed to suitable for the process names (1.jpg, 2.jpg, etc.). Therefore, another text file path, containing the correspondence between the original names of the images and the renamed images, is returned by the concept detection service. The user should copy these two paths and add them to the image clustering URL query. The kmeans clustering algorithm is then executed and the final result is retuned back to the user's browser in XML format. The output presents the N clusters (the number of clusters is user defined) and the corresponding images within each cluster. Also, the dominant image of each cluster is recorded in the XML. A summary of the technical details of the image clustering web service is shown in Table 13.

Table 13: Summary of the technical details of the implemented service

Functional description	Image clustering of images collections
Input	Two text files - One containing a model vector representation for each image of the collection and another for the names of the images
Output	A text and XML file containing the constructed clusters
Language/technologies	JAVA Web Service
Hardware Requirements	N/A
OS Requirements	Windows

The URL query which call the service is: `http://multimedia.iti.gr:8080/ForgetITImageCClusteringService/ImageConceptDetection?pathType=''&pathName=''&renamePath=''&NumberOfCluster=''`

where, **pathType** is the type of the path that the input data will be provided, only the typeUrl type is available now, **pathName** is the path of text file location containing the model vectors (as already mentioned this path is provided by calling the concept detection Web Service), **renamePath** is the path of the text file location containing the image name correspondences and **NumberOfCluster** is an integer indicating the number of clusters that the user want to construct from his/her collection.

An example of the URL query is: `http://multimedia.iti.gr:8080/ForgetITImageClusteringService/ImageConceptDetection?pathType=typeUrl&pathName=http://multimedia.iti.gr:8080/CERTH_BIN/ForgetIT_Concept_Detection_Service/Results/Concepts/VisitEdinburgh2013Sven_user_817_finalDoC.txt&renamePath=http://multimedia.iti.gr:8080/CERTH_BIN/ForgetIT_Concept_Detection_Service/Images/user_817/VisitEdinburgh2013Sven_user_817_RenameList.txt&NumberOfCluster=7`

where an image collection, called "VisitEdinburgh2013Sven", is passed as an input to our concept detection approach. 'VisitEdinburgh2013Sven_ user_ 817_ finalDoC.txt" and

“VisitEdinburgh2013Sven_ user_ 817_ RenameList.txt”, both provided by the concept detection service, contain the image model vectors and the correspondence of the original and renamed image names respectively. Calling the service 7 clusters are constructed and each image of the collection is assigned to one of the 7 clusters. In Figure 14, a representative part of the XML output, which is returned at the user’s browser, is shown.

```
<Image_Clustering>
    <Rename_image_list>
        <Renamed_image>user_817\Visit Edinburgh 2013 - Sven\082.JPG -- 1.JPG</Renamed_image>
        <Renamed_image>user_817\Visit Edinburgh 2013 - Sven\064.JPG -- 2.JPG</Renamed_image>
                        .....
        <Renamed_image>user_817\Visit Edinburgh 2013 - Sven\039.JPG -- 47.JPG</Renamed_image>
        <Renamed_image>user_817\Visit Edinburgh 2013 - Sven\040.JPG -- 48.JPG</Renamed_image>
    </Rename_image_list
    <Cluster_list>
        <Cluster id="1">
            <Clustered_images>2 3 10 21 22 25 26</Clustered_images>
            <Dominant_image>21</Dominant_image>
        </Cluster>
                        .....
        <Cluster id="7">
            <Clustered_images>1 5 6 7 8 23 30 45 48</Clustered_images>
            <Dominant_image>48</Dominant_image
        </Cluster>
    </Cluster_list>
</Image_Clustering>
```

Figure 14: Part of the XML output file

8.5 Conclusions and future work

We studied image/video keyframe clustering based on different types of input data features and clustering methods. While most of the existing methods in the literature use visual features for image/video keyframe representation, we employed model vector representations using concept detection confidence scores. Experiments showed that the kmeans algorithm combined with model feature vectors gives the best result. An open issue that has to be studied thoroughly is the selection of the appropriate number of clusters which in our current work is provided manually. Additionally, the effect of the concept set that is used to the clustering effectiveness is worth studying.

9 Conclusions and future work

In this document the first release of the ForgetIT text and visual information analysis techniques for condensation and summarization was presented, based on the theories and state of the art described in the previous work D4.1 [1]. Several software components performing textual and visual analysis were designed, implemented and evaluated.

In the second year of the project, the aforementioned methods will be extended taking also into account the requirements that will be collected from WP2 (foundations of forgetting and remembering) as well as the application workpackages which have to do with personal (WP9) and organizational (WP10) preservation. Furthermore, these and subsequent ForgetIT methods will continue to be evaluated, initially on ForgetIT datasets and later on also by participation on international benchmark activities. Moreover, during the second project year, the presented methods (implemented in the corresponding software components) will be also tested within the Preserve-or-Forget platform, experimenting the overall processing workflow and component communications (WP5 and WP8).

References

[1] ForgetIT. D4.1 - Information Analysis, Consolidation and Concentration for Preservation - State of the Art and Approach. July 2013. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP4_D4.1.pdf.

[2] ForgetIT. D9.1 - Application Use Cases Requirements Document. July 2013. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP9_WP10_D9.1.pdf.

[3] ForgetIT. D8.1 - Integration Plan and Architectural Approach. October 2013. http://www.forgetit-project.eu/fileadmin/fm-dam/deliverables/ForgetIT_WP8_D8.1.pdf.

[4] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. 2011.

[5] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate Detection using Shallow Text Features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, 2010.

[6] BoilerPipe plugin. [Online; accessed 27-January-2014]. http://gate.ac.uk/userguide/sec:misc-creole:boilerpipe.

[7] Stefan Dietze, Diana Maynard, Nina Tahmasebi, Yannis Stavrakas, Vassilis Plachouras, Elena Demidova, Jonathon Hare, David Dupplaw, Adam Funk, Wim Peters, and Patrick Siehndel. Extraction and Enrichment. Deliverable D3.2, ARCOMEM, 2012.

[8] Termraider tools. [Online; accessed 27-January-2014]. http://gate.ac.uk/userguide/sec:creole:termraider.

[9] Tag Cloud - Wikipedia. [Online; accessed 27-January-2014]. http://en.wikipedia.org/wiki/Tag_cloud.

[10] GATE last build. [Online; accessed 27-January-2014]. http://jenkins.gate.ac.uk/job/GATE-Nightly/lastSuccessfulBuild/.

[11] Christiane Fellbaum, editor. *WordNet - An Electronic Lexical Database*. MIT Press, 1998.

[12] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[13] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[14] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[15] CKEditor Website. [Online; accessed 27-January-2014]. http://http://ckeditor.com/.

[16] Stanford CoreNLP. [Online; accessed 27-January-2014]. http://nlp.stanford.edu/software/corenlp.shtml.

[17] Apache OpenNLP. [Online; accessed 27-January-2014]. http://opennlp.apache.org.

[18] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1535–1545, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[19] Raman Maini and Himanshu Aggarwal. Study and comparison of various image edge detection techniques. *International Journal of Image Processing (IJIP)*, 3(1):1–11, 2009.

[20] Frederic Jurie and Bill Triggs. Creating efficient codebooks for visual recognition. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 604–610. IEEE, 2005.

[21] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[22] Koen EA Van De Sande, Theo Gevers, and Cees GM Snoek. Evaluating color descriptors for object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1582–1596, 2010.

[23] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[24] Engin Tola, Vincent Lepetit, and Pascal Fua. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(5):815–830, 2010.

[25] Jan C van Gemert, Cor J Veenman, Arnold WM Smeulders, and J-M Geusebroek. Visual word ambiguity. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(7):1271–1283, 2010.

[26] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010.

[27] Gabriela Csurka and Florent Perronnin. Fisher vectors: Beyond bag-of-visual-words image representations. In *Computer Vision, Imaging and Computer Graphics. Theory and Applications*, pages 28–42. Springer, 2011.

[28] Vladimir N Vapnik. Statistical learning theory. 1998.

[29] OpenCV. [Online; accessed 27-January-2014]. http://opencv.org.

[30] G. Bradski. *Dr. Dobb's Journal of Software Tools*.

[31] Paul Over, George Awad, Martial Michel, Jon Fiscus, Barbara Shaw, Wessel Kraaij, Alan F Smeaton, and Georges Quénot. TRECVID 2012 - An overview of the goals, tasks, data, evaluation mechanisms and metrics. In *TRECVID 2012*, 2012.

[32] CUDA. [Online; accessed 27-January-2014]. http://www.nvidia.com/object/cuda_home_new.html.

[33] Emine Yilmaz, Evangelos Kanoulas, and Javed A Aslam. A simple and efficient sampling method for estimating AP and NDCG. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 603–610. ACM, 2008.

[34] Eli Peli. Contrast in complex images. *JOSA A*, 7(10):2032–2040, 1990.

[35] Anush Krishna Moorthy and Alan Conrad Bovik. A two-step framework for constructing blind image quality indices. *Signal Processing Letters, IEEE*, 17(5):513–516, 2010.

[36] A. Mittal, A.K. Moorthy, and A.C. Bovik. No-reference image quality assessment in the spatial domain. *Transactions on Image Processing, IEEE*, 21:4695–4708, 2012.

[37] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[38] A.C. Gallagher and Tsuhan Chen. Clothing cosegmentation for recognizing people. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[39] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In Bernd Michaelis and Gerald Krell, editors, *Pattern Recognition*, volume 2781 of *Lecture Notes in Computer Science*, pages 297–304. Springer Berlin Heidelberg, 2003.

[40] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987, 2002.

[41] M. Castrillon, O. Deniz, C. Guerra, and M. Hernandez. Encara2: Real-time detection of multiple faces at different resolutions in video streams. *Journal of Visual Communication and Image Representation*, 18(2):130 – 140, 2007.

[42] MATLAB. *version 8.1 (R2013a)*. The MathWorks Inc., Natick, Massachusetts, 2013.

[43] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[44] Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[45] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[46] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. Wiley. com, 2009.

[47] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[48] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[49] Alexander Strehl and Joydeep Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3:583–617, 2003.

[50] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *Computer Vision–ECCV 2010*, pages 143–156. Springer, 2010.