UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Automatic Image Tagging based on Context Information

Martin Hætta Evertsen

INF-3981

Master's Thesis in Computer Science

June, 2010

Abstract

People love to take images, but are not so willing to annotate the images afterwards with relevant tags. Manually tagging images is both subjective (dependent on annotator) and time consuming. It would be nice if the tagging process could be done automatically. A requirement for effective searching and retrieval of images in rapid growing online image databases is that each image has accurate and useful annotation.

This thesis shows that automatic tagging of images with relevant tags is possible by using a combination of the capture location, the date/time when the image was captured and an image category. The use of image categories (together with location and date/time) ensures that many relevant tags are returned and restrict the occurrence of noisy tags to a very low level despite using a noisy image database (Flickr). Other methods used for further restricting noise are to restrict usage of more than one image from same user (as basis for tagging the query image) and a dynamic approach for using many images when possible, and fewer images when not many relevant images are found.

The designed system is able to tag an image as long as there are a sufficient number of geo-referenced and already tagged images that is relevant for the query image available on Flickr. The query image must also have been geo-referenced and it is assumed that the user provides an image category. Images are processed based on which category the images belongs to, i.e. an image is processed with the best method to handle images belonging to that specific category. In short, this means that images of objects or places are processed differently than images from events.

The evaluation of the system indicates that usage of image categories is very helpful when tagging images. The system finds more relevant tags and fewer noisy tags than baseline systems using only location. It also performs good compared to a system using both location and content-based image analysis.

Acknowledgements

First of all, I would like to thank my supervisor, associate professor Randi Karlsen, for the idea of this work and for very constructive guidance along the way. Your availability, feedback and knowledge are very much appreciated.

I would also like to thank Jan Fuglesteg for help with administrative and practical issues. Many thanks also to family and friends for support and motivation.

Finally, a special thanks to my fellow student David Sundby. We have studied, collaborated, struggled, discussed, laughed, partied and had fun all these five years. Thank you, and good luck with little Theo!

Contents

List of tables, figures and images

Chapter 1

Introduction

This chapter is an introduction to the thesis that will discuss the motivation behind the work, the problem and contribution, a quick overview of what has been done, the assumptions, some basic terminology and finally an overview of the organization of the thesis.

1.1 Motivation

Whereas in the past a roll of film was sent in to a professional photographer to get images developed into handheld photographs that were later glued into photo albums, today most or practically all images are taken and stored digitally. People capture images at an ever-growing rate. It is driven by the development in storage and capture devices (such as digital cameras and camera phones). The cost for these capture devices and storage keeps decreasing, and is affordable for the average person.

Handling large volumes of digital information becomes vital as online resources and their usage continuously grows at high speed. Online image sharing applications are getting extremely popular. Flickr[1] is one of the most popular of these applications hosting over 4 billion[2] images. Over 100 million of these images are geo-referenced. Flickr reports that currently more than 2 million geo-referenced images are uploaded every month. Panoramio[3] is another popular image sharing website where all of the over 10 million images are geo-referenced. Flickr and Panoramio have been bought by the well-known and powerful companies Yahoo![4] and Google[5] respectively,

[1] http://www.flickr.com/
[2] http://blog.flickr.net/en/2009/10/12/4000000000/
[3] http://www.panoramio.com/
[4] http://www.yahoo.com/
[5] http://www.google.com/

which shows the potential of these online image collections.

Datta et al. [1] performed a test using Google Scholar[6] that indicated an exponential growth in image retrieval and closely related topics during the period 1995 - 2005. This increase in image related topics seems to have continued over the last five years.

Manually tagging images is both time consuming and subjective. People simply do not bother or have time to tag their images. Furthermore, human beings are and think differently, meaning that similar images will be tagged differently by different people. This can be caused by differences in language, mood, vocabulary, education, culture, taste etc.

Some of the digital cameras on the market today already have built-in GPS (for example Panasonic TZ10[7]), and the number is increasing. There also exist solutions where a GPS receiver is attached to the flash connector of digital cameras. The images are then geo-coded when they are transferred from the digital camera to a computer with Internet connection. Furthermore, several of the mobile telephones on the market today (for example Nokia N-95 and iPhone 3GS) are equipped with both accurate GPS systems and cameras able to take images with high quality.

There is also a significant increase in tools and applications for manually geo-referencing images. The usage is often very simple; the users drag and drop their images to the position on the map where the image was taken. These applications can show maps where users can see where their own images were taken and also images from other users. Flickr and Panoramio offer built-in geo-referencing using Yahoo Maps and Google Maps respectively. Actually, it is a requirement in Panoramio because all of its images have to be geo-referenced.

Thus, it is very likely that a lot of images in near future will have GPS coordinates available, generated either automatically or manually. The location where an image is taken can be a very valuable asset when tagging images. Location can be combined with other contextual information sources such as weather information, nearby buildings and facilities, date/time (in case of an event taking place), other images taken nearby and geo-referenced articles. This information can for example be helpful when automatically tagging images.

The increase in digital images and research in image related topics together with the problems concerning manually tagging of images indicates that there is a need for automatic image tagging.

[6] http://scholar.google.com/
[7] http://panasonic.net/avc/lumix/compact/zs7_tz10/functions.html

1.2 Problem description and contribution

The specific goal of the work in this thesis is to design, implement and evaluate a system that automatically finds tags for images based on location (GPS coordinates), date/time and image category. The tags are to be collected from an online image sharing database (Flickr) with images that are already tagged. However, the information in these community based collections is often highly unreliable and noisy. Therefore, the thesis will further focus on the relevancy of the collected tags, and how this is affected by using a combination of the context sources (location, date/time and image category) as input to the system.

Location is generally a widely used context source, and there exist several location based image tagging systems (discussed in related work in Chapter 3). However, as far as I know, no previous work has looked into the possibility of combining location with image categories and date/time. The suggested approach is to handle similar types of images (belonging to the same image category) in a specific way giving the best results for that specific type of image.

The contribution of this work is to explore the possibility of making an automatic image tagging system based on combining category, location and date/time. Further, the most interesting aspect of this work will be to evaluate if it is beneficial for an automatic image tagging system to handle images differently based on which image category the images belong to. The hypothesis is that using categories together with location and date/time will result in more relevant and less non-relevant tags than by using other approaches.

This thesis is part of the CAIM[8] (Context Aware Image Management) project. CAIM is a research project with the goal of developing methods and tools for context aware image management in distributed, multimodal and mobile environments. The project is a collaboration between the University of Tromsø, the University of Bergen, NTNU[9] and Telenor R&D[10].

1.3 Categories

Location and date/time information is assumed to be available in the EXIF-header of the image. Category is not. The idea is that users should provide the image categories along with the query images (the images that are to be tagged).

[8] http://caim.uib.no/index.shtml
[9] NTNU (Norges Teknisk-Naturvitenskapelig Universitet) (http://www.ntnu.no/)
[10] Telenor Research & Development (http://www.telenor.com/rd/)

In another work in the CAIM project named InfoAlbum [2, 3], useful information related to an image is found based on category, location and date/time. The system use external sources on Internet for finding relevant information by using a mixture of category, location and time as input. Weather information and relevant articles are some of the information the system can find for images. The category is used (1) to determine how information sources are searched, (2) as a keyword when searching the Internet, and (3) to rank the collected information. Some typically used categories in InfoAlbum are *concert*, *tower*, *church* and *football match*.

We think that users will take the effort of providing image categories if that can provide useful and valuable information to the images. As long as the user interface for the categorization process is rather structured and simple, the effort in choosing categories should be affordable. An important aspect is that categories are to be re-used, i.e. several images can fit into each category.

User studies [4] have shown that in general people are willing to devote some effort and time to annotate and tag images with the motivation of making them more accessible for image retrieval. However, users are not likely to be willing to devote enough time to tag all their images or devote enough time to find many relevant and descriptive tags for each image. It is therefore possible that users instead would prefer to only provide image categories to their images if an image tagging system is able to find a set of relevant tags based on the image category (in combination with other available information such as date/time and location).

Another aspect is that people do not necessarily remember the names of all the attractions they have visited. But with categories, it is sufficient to know that the image is taken for example of a *tower* or a *church* or at a *concert* or a *football match* (which should be fairly obvious by looking at the content of the image).

The work in this thesis will try to tag query images based on their image category combined with the location and time of image capture. The main categories used in the system are basically *objects* and *events*. The suggestion is that user-defined categories such as *tower* and *church* belong to the main category *object* whereas user-defined categories such as *concert* and *football match* belong to the main category *event*.

1.4 Approach

The automatic image tagging system designed and implemented in this thesis can basically be divided into two parts. The first part consists of retrieving a set of images that are considered to be relevant for the query image (the image that is to be tagged). The second part consists of collecting and processing the tags of the images in the image set found in the first part.

The images used as basis for tagging the query image are retrieved by sending search requests to Flickr. An important part of the implementation is a dynamic method for deciding how many images to use as basis for tagging the query image. The method ensures that many images are used when many relevant images are available and that few images are used when few relevant images are available. The method basically consists of starting with a small search radius and then increase the search radius until enough images are found.

Categories are used to specify the search requests so that mostly relevant images are used as basis for tagging the query image (i.e. the occurrence of non-relevant images is restricted). This is achieved by using different search parameters for the different main categories. The main categories used in the system are mainly *object* and *event*. One of the differences is that date/time is used as search parameter for *events* but not for *objects*. Another difference is the usage of the user-defined categories in the search process.

The second part consists of processing (sorting, handling whitespaces, filtering etc.) and using the tags of the images found in the first part to tag the query image. Basically, the tags that appear most frequently in the set of images found in the first part are used to tag the query image.

The approach will be discussed in more detail in Chapter 5.

1.5 Assumptions

The assumptions are as following:

- It is assumed that there exist a representative set of already geo-referenced and tagged images on Flickr. The tags of these images are used to tag the query images.
- It is assumed that the user specifies an image category along with the query image that is to be tagged.
- It is assumed that the location where the image was captured is available in the EXIF-header of the query image in the form of GPS coordinates.
- It is assumed that the date and time of image capture is available in the EXIF-header of the query image.

The assumptions will be discussed in more detail in Chapter 6.6.

1.6 Image copyrights

All images used to test and evaluate this work are under a Creative Commons[11] license which allows usage, adaptation (editing) and distribution as long as the work is attributed. The source of the images used to test and evaluate this work can be found in Appendix A. GPS coordinates and date/time information have been added to the EXIF-header of these images. Therefore, the information might not correlate to where and when the images were actually taken.

1.7 Terminology

A *tag* is a keyword or term assigned to an image that helps describe the image and its content so that it can easily be retrieved when searching or browsing for it. Image *tagging* is the process of assigning tags to an image. Image *annotation* is the process of annotating images with relevant information. Tags consists of one or two words (e.g. “London” and “Eiffel Tower”), whereas annotations often consists of several words combined into descriptive sentences. *Automatic image tagging* is the process performed by a computer system of automatically assigning tags to an image.

The *query image* is the image that is to be tagged by the automatic image tagging system. The *user* of the automatic image tagging system is a person who has a query image that he or she wants to tag with relevant tags. The *result set* is the set of returned images that is considered relevant to the query image. Tags from images in the result set are used to tag the query image. *Required images* are the number of images required in the result set. This number can vary depending on how many relevant images that are found for a specific query image.

Noisy tags are tags that are assigned to an image and that is not relevant for the image. An example is if the tag “castle” is assigned to an image of a church. It is common (although not used in this work) to use a *training set*, which is a small and controlled set of correctly tagged images that are used to train a system using machine learning techniques. The noise level in the training sets are very low compared to the relatively high noise level in community image sharing applications such as Flickr. The work in this thesis will use images from Flickr and not a *training set*.

A *geo-referenced* image has the geographic location where the image was captured available. A *geo-coded* image has the geographic location stored internal in the EXIF-header of the image, whereas a *geo-tagged* image has it stored with its external tags.

[11] http://creativecommons.org/licenses/

1.8 Organization

The rest of this thesis is organized as follows. Some useful background information is presented in Chapter 2, and related work follows in Chapter 3. The problem and some limitations are discussed in more detail in Chapter 4. The selected approaches are discussed in Chapter 5, and the overall design is presented in Chapter 6. Some implementation specific details can be found in Chapter 7. The results and evaluation of the results is presented and discussed in Chapter 8. Some possible future work is discussed in Chapter 9, before the conclusion in Chapter 10. The implementation code can be found in the appendices behind the list of references.

Chapter 2

Background

In this chapter some background information relevant for the thesis is presented and discussed.

2.1 Context

A commonly used definition for context is suggested by Dey [5]: "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*". In the work of this thesis, the entity is an image. In other words, the context of an image is all relevant surrounding situational environment and information that describes or belongs to that specific image.

Only relevant information can be regarded as context, i.e. information that is usable for some purpose. But almost all kind of information can be used in some way. The key is to use the information that will assist most in achieving the desired goal. For images, it could be important to know where, when and in which situation the image was taken or what the main subject of the image is.

Images taken by digital cameras store a lot of contextual information in the EXIF-record of the image. This record contains information such as lens settings, focal time, scene brightness, exposure time, camera information and settings, whether the flash was fired or not, the time and date the image was captured and possibly GPS coordinates if the camera is equipped with the Global Positioning System[12].

[12] http://www.gps.gov/

2.2 Image retrieval

The main reason to tag images is for improving the usability of image retrieval systems. Image retrieval systems can be used to find images of interest in both large online public image databases (such as Flickr and Panoramio), and personal image collections stored on home computers. Images that are tagged with a set of relevant tags are easier to find than untagged images. Therefore, image tagging is vital for being able to manage and search effectively in large image collections.

Current image retrieval systems such as Google Image Search[13] and Microsoft Bing Image Search[14] are based on the text surrounding images. While this approach is successful for text retrieval, it is unfortunately not the case with image retrieval. Looking at the surrounding text of an image is not a good approach because it is difficult to know whether and which parts of the text that are relevant for a given image.

Another image retrieval approach is query-by-example or content-based image retrieval [6]. With this approach, images are retrieved based on either an example image or drawing, where the content of the example image is used to find similar images with visual similarities or features such as colors, shapes and textures. However, this approach requires the user to supply an example image, which is not very convenient. Also, it is not useful in cases where the user wants to find all images from a specific event or object where the content of the image can vary a lot. Furthermore, people are familiar with the regular way of searching ("*googling*") on the Internet by typing in one or more keywords to a search engine, and there is no reason to believe that it will be any different with images.

Manual image tagging is both time consuming and subjective. If two people are to tag the same image, it is almost certain that the two individuals will not use the same set of tags. Even the same person is likely to tag the same image differently if being asked to do so with a significant period in between. Moreover, the effort and time required to tag all images manually is too demanding. Hiring people to tag images is not realistic either, as it would imply many thousands of work hours.

To summarize, it is fundamental for image retrieval systems to have the images tagged, and doing it manually is not a realistic option. There is a need for images to be automatically tagged.

---

[13] http://images.google.com/
[14] http://www.bing.com/images/

2.3 Automatic image tagging

The idea with automatic image tagging is that tags are automatically captioned and assigned to the digital image. These tags should describe every important part or aspect of the image and its context. Automatic image tagging can be done based on the visual content of the image, contextual information, or using a mixture of these two approaches.

By looking at the visual content of an image, it could for example be possibly to predict that an image where most edges are vertical or horizontal contains a building. Another approach is to find a set of images that are visually similar to the query image in existing image databases consisting of already tagged images, and then pick the most relevant tags from the set of similar images [7, 8].

The context in which the image was taken can also be used to tag images [9-11]. Context is as discussed in Chapter 2.1 all relevant surrounding situational environment and information. Location and date/time are the most commonly used context sources for most context-aware applications.

The key task for an automatic image tagging system is to be able to tag the query image with relevant tags. The problem is that it is hard for a system knowing exactly what is relevant and what is not relevant for a specific image. Therefore, the results are strictly speaking predictions, and the focus should be on making the predictions as good as possible. The main challenge with automatic image tagging is that many images are difficult to describe in words, particularly those where feelings are involved. The problem is known as the so-called semantic gap.

2.4 Semantic gap

A semantic gap occurs when human observations or behavior are transferred to computational representations. Smeulders et al. [12] defines the semantic gap: "*The semantic gap is the lack of coincidence between the information that one can extract from the visual data and the interpretation that the same data have for a user in a given situation*".

In other words, there is a lack of correlation between the way humans understand information and the way computers represent the same information. Whereas words and text usually have a clear semantic meaning, the same is not the case with images. Analyzing images requires reflective thinking that computers are not capable of, not today and not in foreseeable future.

Textual annotations are unreliable as they depend on attributes of the annotator such as knowledge, culture and language. This makes automatic image tagging challenging because computers and computer systems in general have limited knowledge and limited capability of expression. They lack the ability to think, reflect and learn[15] upon ideas and previous experience. This is in big contrast to humans, where mature and adult people often are better decision makers than young people because they take use of knowledge and experience acquired during their lifetime.

Closing the semantic gap is one of the main challenges in image related topics, and a lot of research has been devoted to the problem [1, 6, 12]. This work (and all other image tagging systems) indirectly tries to minimize the gap by finding as many relevant tags as possible to query images.

2.5 Category

The idea of structuring things in categories (categorization) is that a set of things that are somewhat similar or share some specific characteristics are grouped into the same category. This way of representing things is fundamental in everyday situations, for educational purposes, in research, for companies to structure their products and in many other areas and situations.

There are many different ways to categorize. A common organization is to organize objects into categories and sub-categories. People naturally categorize objects into basic categories such as vehicles or tables based on characteristics such as visual appearance, movement and attributes [13].

For the purpose of this work, images are to be categorized such that all the images in one category can be handled equally. In other words, the image tagging process is performed the same way for all images in one image category. Images in another image category is handled another way, which is the best approach for that specific category.

The game of football can be used as an example. It could mean the ball used to play football with (the object), or the event of a football match taking place at a stadium. Furthermore, someone could visit a football stadium without a match taking place. It is possible that these three different cases should be handled differently by an automatic image tagging system to achieve the best possible results. This can be achieved if the images are placed into different categories (such as object, event or place).

2.6 Location

Location information is likely to be well known information for more and

[15] Although machine learning is an interesting area of research, it will not be discussed in this work.

more images in the future because of the increase in digital cameras with built-in GPS and the possibility to geo-reference images manually. Information about where an image was captured can be very useful when tagging images. For any image with capture location available, useful information can be found by combining location with other information sources such as weather databases, nearby buildings and facilities, time (in case of an event taking place), other geo-referenced images taken nearby and geo-referenced articles on the Internet [2, 3].

Geo-referenced images taken nearby can be found using both Flickr and Panoramio. Flickr is used by the automatic image tagging system designed in this thesis. The usage will be more discussed in Chapter 5 and 6.

GeoNames[16] is an online geographical database available free of charge containing over eight million geographical names updated by users using a simple wiki interface. One of the web services they offer return nearby location names based on GPS coordinates. This is called reversed geo-coding, and is very useful for automatic image tagging because the location names themselves are often useful tags. An image taken in Paris could be tagged with location names spanning all the way from Europe and down to the name of the neighborhood or even street name in Paris.

Further, it gives the opportunity to search for useful information concerning the location where the image was taken. For example, another service based on GPS coordinates offered by GeoNames, finds nearby geo-referenced Wikipedia[17] articles[18]. The articles can provide useful information about the subject of the image. For example, for an image taken next to the Eiffel Tower, the Wikipedia article about the tower could provide useful information.

2.7 Synonyms

Another interesting issue concerning image tagging is handling synonyms. For example, it should be obvious that train and railroad are two words for the same thing. Images tagged with train could therefore possible be tagged with railroad as well. Another possibility is to use it the other way around, meaning that image retrieval systems searching for images with trains will use both *train* and *railroad* in the search request.

STANDS$_4$ LLC[19] is a leading provider of free online reference resources offering a simple API for retrieving synonyms for English words. WordNet[20]

[16] GeoNames (http://www.geonames.org/)
[17] Wikipedia – The Free Encyclopedia (http://en.wikipedia.org/)
[18] http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Geographical_coordinates/
[19] http://www.abbreviations.com/about.aspx
[20] http://wordnet.princeton.edu/

[14] is a large more advanced lexical English database where nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms.

2.8 Relevancy

Tagging images is subjective, and the same applies to deciding whether tags are relevant or not for a specific image. Whereas one person might consider a tag relevant for an image, another person might consider the same tag as noisy for the same image. There are many tags that are potentially hard to decide whether are relevant or not. An example is information relevant for the position of image capture but that is not visible on the image. Another example is whether tags that do not give any meaning on their own without being combined should be regarded as relevant (e.g. *big* and *ben* for an image of Big Ben).

Precision and recall are two widely used statistical classifications for measuring relevancy in information retrieval systems. It can also be used to measure relevancy of tags:

- Precision is the fraction of tags that are relevant. In other words, it is the number of relevant tags found (for an image by an image tagging system) divided by total number of tags found for the image by the image tagging system. A perfect precision score of 1.0 means that all tags returned are relevant.

- Recall is the fraction of tags that are relevant divided by the total number of relevant tags that should have been found. A perfect recall precision score of 1.0 means that all relevant tags that could be found was found.

Chapter 3

Related Work

In this chapter some related work and their difference to the work in this thesis will be discussed. The related work is grouped into location-based, category-based, time-based and visual similarity-based image tagging.

3.1 Location

SpiritTagger[21] [7] is a geo-aware tag suggestion tool using Flickr that suggests geographically relevant tags for images with GPS coordinates. It does so by combining the geographical context with content-based image analysis. Geographic mining is done by collecting a set of images that are within a certain radius of the candidate image to be tagged. This set of images is narrowed down by using visual similarity techniques. The tags of the images in the set are then compared to their global frequency. Local frequency refers to the frequency of a tag in the result set, whereas global frequency refers to the frequency of a tag in all images on Flickr. Tags with higher local frequency than global frequency are assumed to be relevant for the query image. Experiments shows that SpiritTagger works well compared to baseline methods that only use geographical context. The work of SpiritTagger is related with the work in this thesis in that uses Flickr for finding relevant images. Further, it uses location to find nearby and possibly relevant images, but whereas SpiritTagger use content-based image analysis to narrow down the set of images, the work in this thesis will use a combination of category and date/time. SpiritTagger is used in the evaluation (Chapter 8) for comparison against the system designed and implemented in this thesis.

MonuAnno [15] automatically annotates landmark images. They refer to landmarks as geographically situated objects or small areas such as Eiffel Tower and Big Ben. An important part of the system is a reference database of landmarks generated based on image locations and visual similarity from

[21] http://cortina.ece.ucsb.edu/index.php

images on Flickr and Panoramio. The annotation of a query image consists of two steps. The first step is to decide which of the nearest landmarks the query image belongs to. The second step is to verify that the query image indeed belongs to that landmark. Visual similarity and location is used in both steps. Whereas they use a reference database of landmarks, the work in this thesis will use categories. Further, the work in this thesis will not only support landmark images. Finally, whereas MonuAnno only tag with the name of the landmark, the focus of this work will be to find a set of relevant tags for a query image.

ZoneTag [10] is a mobile phone application allowing and encouraging users to easily upload images taken with their mobile phones directly to Flickr at the time of capture. It also suggests tags based on context information such as previously used tags and names on nearby attractions. The client (mobile phone) communicates with a server that performs computational and time-consuming tasks unsuitable for mobile platforms. It differs from the work in this thesis mainly in that ZoneTag does not use categories. Further, location information is only approximated if exact GPS location is not available. Also, ZoneTag is a mobile application that uploads images as they are taken.

3.2 Category

AnnoSearch [8] is a system that annotates images based on search using a keyword and the image itself. First, a text-based web search is performed to find a set of semantically similar images. AnnoSearch then use the query image to find a set of visually similar images. Next, the two set of images are clustered into sets of keywords (for example *castle*, *cloud and tree*). Finally, these keywords are ranked according to frequency and visual similarity to the query image. The top ranked keywords are assigned as tags to the query image. Experiments on 2.4 million images proved the effectiveness and efficiency of the system. AnnoSearch is related with the work in this thesis in that it use and require a keyword/category from the user, but while AnnoSearch combines it with the use of visual similarity techniques, the work in this thesis combines it with location and date/time.

Rattenbury et al. [16] shows that it is possible to determine whether an existing tag on Flickr represents an event or a place. They demonstrate that if a certain tag represents an event or a place, then that tag must have a significant higher frequency in a certain time scale and/or in a certain area compared to its general frequency outside this time scale or area. Whereas they in [16] categorize existing tags from already tagged images on Flickr based on location and date/time, the work in this thesis will focus on the process of tagging un-tagged query images by using the location, date/time and user-defined categories. Thus, the work in [16] automatically *categorize existing tags* (not images) whereas the work in this thesis will automatically *tag images based on given image categories*. However, the work in [16] proves that it is possible to identify different categories on Flickr. Further, and outside the scope of this work, it could be possible to build on the work in [16] to automatically categorize images (discussed more in future work in Chapter 9).

The problem of image classification (categorizing images) has been looked into by some other works, especially in relation to machine learning techniques. The work in [17] looks at how the two problems of image classification and image annotation can be connected. Their method consist of using content-based image analysis, machine learning and probabilistic models on a training set to automatically classify images and annotate images. Their work indicates that categorizing images is supportive when annotating images. However, they do not take use of location information nor date and time information. Further, they use a training set with around 200 images for each class/category with no or very little noise whereas this work will use an existing and real image database where noise must be considered as normal rather than as exception.

3.3 Time

Date and time in itself is often not enough to base automatic image tagging systems on. But it can be very useful when used in addition to other approaches.

The work in [18] indicate that people often tag their images with when it was taken (in addition to where it was taken and the content of the image). User studies in [19] demonstrate that finding images on image browsers using date and time are significantly faster compared to finding the images in image browsers not taking advantage of date and time information.

The work in [16] use time to detect events. A certain tag could represent an event if that tag has a significant higher frequency in a limited time scale compared to its general frequency outside the limited time scale.

The work in [20] found that in general, they believe that fewer people georeference images of events. When fewer images are available to work with, it is obvious that it is more difficult to find relevant tags. This seems to agree with the general consensus in the field of image retrieval and image annotation; that handling events is a difficult task.

3.4 Visual similarity

For systems using content-based image analysis / visual similarity techniques (e.g. SpiritTagger, MonuAnno and AnnoSearch), non-relevant images can be discarded based on their low similarity score compared to the query image. On the same basis, images that get low similarity score compared to the majority of the returned images (or the normal of the returned image set) are also likely to be non-relevant. This is an advantage compared to systems not using visual similarity because it is harder to discard images that are not relevant when the visual content of the image is not analyzed.

However, a severe problem with the content-based image analysis is that images are taken from different views and angles. This makes it harder to find visual similarity among images of the same attraction. The back side of a building is not necessarily very similar to the front side of the building, and the background can also be significantly different on images taken in opposite directions (or in a different season, time of day etc.). Similarly, images from an event does not necessarily have to be visually similar (consider a concert where images are taken both of artists and spectators). Another related problem is that people often take images of themselves in front of attractions, which will disturb the visual similarity techniques. The approach of the work in this thesis avoids these problems because only metadata information is used to locate relevant images.

3.5 Other related work

In a related master thesis completed in January 2010, Jakobsen [3] proved that it was possible to collect relevant context information related to an image using date/time of image capture, capture location and a user-defined image category. The InfoAlbum prototype demonstrating the information collection is reported in [2]. InfoAlbum utilizes many external sources to acquire information such as location names (reversed geo-coding), weather information (from a weather history database) and location specific articles. These works demonstrate that it is possible to use category, date/time and location to collect relevant information about the image. The work in this thesis will focus on collecting relevant tags to a query image based on related images found by using a combination of category, date/time and location.

Ames and Naaman [4] performed a user study using Flickr and ZoneTag and exposed that the main motivation for tagging images is functionality. People want to tag and organize images to make it easier both for themselves and for others to search, browse and retrieve images. Sigurbjornsson et al. [18] found that users of Flickr tag their images with (1) where the image was taken, (2) who or what is on the image, and (3) when or in which occasion the image was taken. This was found by performing an experiment classifying tags from a set of images on Flickr with the use of the classification system used by WordNet [14].

Kennedy et al. [21] reports that it for any given tag on an image on Flickr is only roughly 50 percent likely that the concept of the tag actually appears on the image. This can be caused by noisy tags or disagreements in concept definition. The latter can be caused by disagreement in whether an image taken from a given building (but where the building itself is not on the image) should be tagged with the name of the building or not. There is also a problem in deciding whether the location name where images are captured (Paris, Greece, London etc.) should be included as tags to query images. And further, which accuracy of location names to use (e.g. Europe and/or Downing Street and/or something in between).

Chapter 4

Problem Description

In this chapter the problem to solve and some of its difficulties and limitations will be presented and discussed in more detail.

4.1 Problem definition

As stated in the introduction, this thesis is part of the CAIM[22] (Context Aware Image Management) project, which is a research project with the goal of developing methods and tools for context aware image management in distributed, multimodal and mobile environments.

The general problem to solve is to automatically tag images. The tagging process will be based on images from Flickr that are already tagged and geo-referenced. Community based image collections like Flickr usually have much noisy information, but is preferred as the collections are big, easily accessible and rapidly growing. This makes them more interesting than training sets made especially for tagging purposes. A combination of metadata / context sources (image category, location given by GPS coordinates and date/time of image capture) is to be used to find relevant images and restrict the occurrence of images that are not relevant. Tags from these images, considered as relevant for the query image, are then to be used as basis for tagging the query image.

Relevant images are to be found by using the image category combined with the location and time of image capture. User-defined categories such as *tower* and *concert* should belong to a main category. The main categories used in the system are basically *objects* and *events*. The main categories should be used to define the search parameters when searching for a set of relevant images. Thus, the search parameters for the different main categories are different. The user-defined categories can be used as one of the

[22] http://caim.uib.no/index.shtml

search parameters. Location and date/time are other possible search parameters.

4.2 Contribution

The contribution of this work is to explore the possibility of making an automatic image tagging system based on combining category, location and date/time information. Further, the work will evaluate whether it is beneficial for an automatic image tagging system to use image categories. The hypothesis is that using categories together with location and date/time will result in more relevant and less non-relevant tags than by using other approaches.

4.3 Scenario

Bob and Alice have just returned from a round trip all over Europe, and have taken a lot of images with their new digital camera. The camera is equipped with a GPS system. After uploading the images from the camera to their computer, they can browse the images based on capture dates. They can also use an application presenting where on a map the images were taken (since the images have been geo-referenced). But they soon realize that there is no real structure on the images. They want to have relevant information assigned to each image about where the image was taken, what the content of the image is and when or in which occasion the image was taken [18]. Further, they want to be able to allow themselves, friends, family and possible everyone to easily find for example all images of castles or all images from concerts [4]. Image tagging is the solution, but they do not have (or take) time to do it manually. In fact, they probably do not remember the names of all the different objects, places and events they have visited.

Instead, they use the automatic image tagging system designed in this thesis which tags images based on three context sources; location, date/time and category. Location and date/time are automatically obtained from the EXIF-header of the query image, while the users must manually specify the image category. The system will then use a combination of the three context sources to find a set of relevant images. Tags are collected from this set of relevant images and used to tag the query image.

Two images, Image 4.1 and Image 4.2, are used as examples. Bob and Alice assign the images the categories *tower* and *concert* respectively. These categories belong to the main categories *object* and *event* respectively. The tags listed in Table 4.3 are the set of tags that the automatic image tagging system in this thesis found for the two query images. Note that there exist more tags that can be considered as relevant, and some of the tags listed might not be considered relevant. Other image tagging systems may find another set of tags, as they collect tags using other techniques and parameters. The relevancy of the tags will be discussed in the evaluation in Chapter 8.

Image 4.1 – Image taken at Westminster Bridge in London showing Big Ben which is part of Houses of Parliament.

Image 4.2 – Image from a U2 concert at Camp Nou in Barcelona during their 360 tour in 2009.

Tags for Image 4.1	Tags for Image 4.2
London	Barcelona
Big Ben	U2
tower	Camp Nou
England	360
clock	Spain
Westminster	concert
UK	Bono
clock tower	tour
United Kingdom	Nou Camp
Parliament	edge
big	The Claw
ben	Catalonia
	Concierto
	Catalunya

Table 4.3 – List of tags found for Image 4.1 and Image 4.2 by the automatic image tagging system designed and implemented in this thesis.

Everyone with access to the their image collection can now easily search for specific images, and will have useful information available when browsing the images without the need for Bob or Alice to guide them and trying to remember the names of the different attractions. Bob and Alice can now happily go and do something useful as the images are both categorized (manually) and tagged (automatically).

4.4 General problems and limitations

Consider an image taken from the same location as Image 4.1 but in the opposite direction. Because the location is still the same, some tags such as London and possibly Westminster Bridge are relevant for both images. But other tags are not. Big Ben is an obvious tag for the original image, but it is not relevant for an image captured in the opposite direction. Instead, other objects are relevant, the most obvious being London Eye, a large passenger-carrying Ferris wheel located on the other side of the River Thames than Big Ben. Two possible approaches for distinguishing the two cases are with use of content-based image analysis or use of image categories. The usage of content-based image analysis has been investigated by others (as discussed in related work in Chapter 3). The work in this thesis will focus on image categories and not use the visual content of the image.

Location is used as one of the search parameters, and therefore only images inside a certain radius is used as basis for tagging the query image. There must therefore be enough images available on Flickr that is relevant for the query image before the system is able to tag the query image with relevant tags. Images taken of a random object will probably not get relevant results because the probability that other users have taken images of the same random object nearby the same location is very low. In fact, it is likely that nearby images are taken of other objects (since the original object is not special, i.e. not an object that many people take image of). However, these objects can possibly be recognized by the content of the image using visual similarity techniques, but that will not be the focus of this work.

The red double-decker bus in Image 4.1 is interesting in that it is an object that many people take images of although it is not location dependent. The bus can occur at several different locations in contrast to Big Ben and many other objects and events that have a fixed location. However, it only exists at certain locations in the world (most famously in London). Regarding Image 4.1, it is difficult to argue that the bus is the main object in the image. However, treating it as the main object will give fairly good results using the system implemented in this thesis. This will be discussed further in the evaluation in Chapter 8.4.9.

The bridge (Westminster Bridge) in Image 4.1 is hard or even impossible to spot on the image. It could be argued that the bridge is of no relevance to the image despite the fact that the image is taken from the bridge. Similarly, it is difficult to tell that the concert in Image 4.2 is at Camp Nou by just looking at the content of the image. Still, the name of the stadium where the group performed is a relevant tag for the image.

In Image 4.2, U2, Bono (main vocalist in U2) and Edge (another member of U2) can possible be detected using face detection or something related. But that would require advanced visual similarity techniques and a big, accurate and up-to-date database of famous people. Further, it is likely to have a high error rate as artists tend to change their image and visual appearance frequently. Also, as can be seen on Image 4.2 it is not easy to spot the faces of the artists. In addition, some event related tags are not detectable from the content of the image. It is for example not possible to see visually that the image was taken from the “360 tour”. Therefore, it might be easier to find this information by handling it as contextual information. The system implemented in this thesis should be able to collect tags for an event as long as there are enough relevant images available on Flickr from the event.

Regarding events in general, it should be easier to find tags for public events because it is likely that there exist more images from these events than it does for private events. Private events are for example weddings, family dinners or birthdays, whereas public events are for example concerts, festivals and football matches. Further, for a private event taking place at some kind of “party house” and/or church, it could exist images from previous private events with other families that are not relevant. It is not desirable to tag a wedding image with wrong names on the newlyweds.

Tags can also be found by combining information from the EXIF-header with external information sources. Weather information can be acquired by specifying location and time to a weather database. Time of day or season can be found by combining location and time using calendars and time zones. Using this approach, Image 4.1 could have been tagged with *sunny*, *hot*, *morning* and *summer*. Similarly, Image 4.2 could have been tagged with *temperate*, *summer* and *evening*. The approach of finding weather information have been explored [2, 3, 9, 11] and will not be implemented in this system.

Similarly, location names can be found by specifying GPS coordinates to GeoNames. Image 4.1 could have been tagged with location names ranging all the way from Europe, United Kingdom, England, London, Central London, Westminster and down to Westminster Bridge Road. Again, this approach has been explored [9, 11, 22]. The biggest problem with location names is to decide which accuracy to use. The work in this thesis will not use GeoNames to find location names, but instead use the location names that exist on related images. The idea is that location names used frequently in related images should be suitable choices for describing the location of the query image.

Chapter 5

Approach

In this chapter the selected approaches are presented and discussed. The first part consists of retrieving a set of images that are considered as relevant for the query image. This will be the subject of Chapter 5.1, 5.2 and 5.3. The second part consists of processing the tags of the images in the result set and is the subject of Chapter 5.3 and 5.4.

5.1 Location

The location of the image is assumed to be available in the EXIF-header of the image in the form of GPS coordinates. The location of an image can refer to the camera position at the moment of capture or the exact location of the subject of the image. Images automatically geo-coded by digital cameras with built-in GPS will use the position of the camera at capture time whereas users manually geo-referencing images often will use the exact position of the subject of the image even if the image was taken from distance to the subject. Thus, both approaches are used, and must be taken into consideration. Further, images can be slightly wrongly geo-referenced because of inaccurate GPS systems or sloppy users.

The location *radius* to use when searching for nearby images is an important aspect of image tagging systems using location. Theoretically, the radius should be as small as possible as long as it covers the area of interest and as long as there are enough images inside the area that are relevant. Practically it is not so easy, however. If the chosen radius is too small, there is a possibility that not enough images will be found inside the radius.

However, there is also a problem with using a too big radius. Imagine an area with two attractions where one of the attractions has a lot more images than the other attraction. This is illustrated in Figure 5.1. The query image is located in the middle of the Figure with the small attraction in the middle. The bigger attraction to the right is not relevant for the small attraction. The dots represent images and as can be seen, the bigger attraction has more images available than the small attraction. The circles on the figure represent three selected radiuses that can be used for retrieving images. With the smallest radius, just a few images are found. Therefore, it is tempting to increase the radius to find more images. But only a few more images are found when the radius is increased. However, when increasing the radius even more, a lot more images are found. But the problem is now that since there are more images from the bigger attraction than the smaller attraction, the automatic image tagging system is likely to select tags that are related to the bigger attraction. In this case, that could lead to selecting wrong tags because the attractions are not related at all.

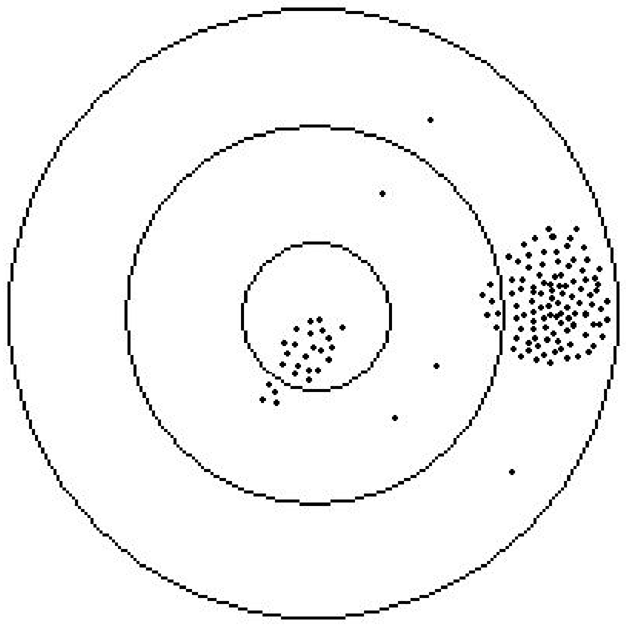


Figure 5.1 - Overview map of two bunches of images and three varying radius sizes. The images are represented by dots and the radiuses are represented with circles. The attraction of interest is the small attraction in the middle of the figure, whereas a big attraction not relevant for the small attraction is located in the right hand side of the figure.

Another related problem is that there are a different amount of relevant images available for each query image. For example, there are a lot of images of famous buildings (e.g. Eiffel Tower and Big Ben) available, and as a result a lot of relevant images are located in a very narrow area close to the actual object. Contrary, events often have fewer images available divided over bigger areas. There are fewer images available because events are time limited in contrast to objects where images captured at any time can be used. The search radius for events must therefore often be increased to find enough relevant images to add to the result set. Further, the area for events is often bigger because an event can occur over a bigger area whereas an object is usually constant placed.

Because of these problems, it was not easy to decide which radius to use and how many images to use (required size on result set). It does not help to increase the result set if the result set is filled with images that are not relevant to the query image. Contrary, it is not desirable to use too few images if there are more images available that are relevant for the query image. This is because using more images will give better and more secure results than using just a few images (as long as the images added to the result set are relevant).

A try and fail approach were used to find a general solution working both for images with many relevant images available and for images with few images available. As expected (due to the varying amount of relevant images), choosing a constant number of required images worked badly. Similarly, using a constant radius was not useful. Further, it was not a good idea to require fewer images for events and more images for objects. This is because some objects only have few relevant images available and some events have many relevant images available.

Instead a more advanced dynamic approach is taken. The goal is to use many images if there are many images available close to the query image, and fewer images if there are not many images located close to the query image. The reasoning is that if many images are geo-referenced very close to the query image, it is likely that these images are relevant, and therefore as many of them as possible should be used. Contrary, if few images are found very close to the query image, it is likely that it does not exist that many images that are relevant for the query image. Therefore, fewer images should be used. But there might exist relevant images that are not located very close to the query image, and these images should be used if there are few relevant images available. These images could have been taken from distance to the object/event, the images could be wrongly geo-referenced or the object/event might cover a big area.

The solution is therefore to start the image search to Flickr with a very small *radius* (0,001 km = 10 m) and a fairly high amount of *images required* (50) in the result set. In the first search using *radius* at 0,001 km, the result set is big enough if 50 or more images are returned. If enough images are returned, then the tags of these images can be processed. If not enough images are found, then the *radius* is doubled and the *images required* in the result set is decreased with one tenth of its last value as shown in Table 5.2 (50 – (50 / 10) = 45). The image search to Flickr continues until the result set have reached the *images required* variable. The maximum allowed radius to use in Flickr is 32, and the search process will therefore end when the radius reaches 32 even if no images are found.

The result set is regarded as big enough the moment it has exactly enough images required in the result. The other possibility would have been to use all images returned in the latest search for tagging the image. However, this was found to be ineffective as it sometimes caused the runtime of the system to be slower since more images were processed (sometimes up to several hundred) without giving notably better results. Thus, it was found to be sufficient to only use exactly the number of required images as shown in Table 5.2.

Radius of image search (km)	Images required in result set
0,001	50
0,002	45
0,004	41
0,008	36
0,016	33
0,032	30
0,064	27
0,128	24
0,256	22
0,512	19
1	17
2	16
4	14
8	13
16	11
32	0

Table 5.2 – A dynamic approach for deciding required size of images in result set. The search for relevant images is performed (starting at the top) until there is enough images in the result set (second column). The radius (first column) is doubled for each search (down the list).

5.2 Category

Location and time is available in the EXIF-header of the image. Category is not. It is assumed that the user should provide a category along with the query image.

5.2.1 Overview of usage

The objective is to handle each of the categories differently such that the result for each category is optimized. In other words, the query image should belong to the category which gives the best tags for that specific type of image. Each category should thus process its set of images in a specific approach, which is the best possible approach for that specific type of image.

Further, the image categories are divided into main categories and user-defined sub-categories:

- The main category is used to specify which search parameters to use when searching for relevant images on Flickr. Basically, the two main categories used are *object* and *event*. Users cannot make new main categories, as they are part of the implementation.

- The sub-category belongs to a main category. Sub-categories such as *castle* and *tower* belong to the main category *object*. Other sub-categories such as *concert* or *Halloween* belong to the main category *event*. Sub-categories are user-defined. Each of the main categories can therefore have many sub-categories.

The user must provide an image category when uploading an image. The image can be placed in one of the existing sub-categories if the image fits into an existing sub-category. If the image does not fit into one of the existing sub-categories, the user can make a new sub-category. To make a new sub-category, the user must specify a name for the sub-category and choose which of the main categories the new sub-category should belong to.

Users cannot make new main categories, as the main categories are part of the system. They are used to define which search parameters to use in the search process. The goal is to select search parameters such that many relevant images are returned and in the same time restrict the occurrence of images that are not relevant. Date/time and sub-category can be used as search parameters. Location is another possible search parameter, but as described in Chapter 5.1 it will handle all images using the same method regardless of category.

Image 4.1 has the sub-category *tower* which belongs to the main category *object* whereas Image 4.2 has the sub-category *concert* which belongs to the main category *event*. Therefore, these images are handled differently by the image tagging system. More specifically, date and time information is used as search parameters for the *concert* image but not for the *tower* image. Another difference is that the sub-category is used as a search parameter for the *tower* image but not for the *concert* image.

The usage of the sub-category as a search parameter can be explained by an example. Image 4.1 has the category *tower*, and therefore images which have *tower* associated with itself (in its tag, title or description) are likely to be relevant for the image. Therefore, tower is used as a search parameter. Thus, only images where the word "*tower*" appear in the metadata information is used by the image tagging system for tagging the query image 4.1.

The usage of date/time as search parameter can also be explained by an example. Image 4.2 has the category *concert* which belongs to the main category *event*. Since events occurs inside a given time interval, only images captured inside the given time interval is used by the image tagging system for tagging the query image 4.2.

A more detailed description concerning the usage of both main and sub-categories will follow in Chapter 5.2.6, but first categorization must be discussed in more detail.

5.2.2 Overview of categorization

An important part of the work is to decide which main categories to use in the system. There are many aspects to consider when categorizing. The overall goal is to choose main categories in such a way that all kinds of situations are covered, and that the results for each of these situations are optimized based on the input and other variables. On the other hand, having too many main categories is not desirable. That would just confuse the user and introduce or increase the error rate because it could lead users into choosing wrong main categories when they create sub-categories. Some users could even just give up and choose a random main category if there are too many choices. Thus, the number of main categories should be restricted.

Another aspect is that the main categories and their names should be easy to understand, meaning that the users should not need to reflect and analyze on what is actually meant with the name of the category. Further, it should be evident whether a certain image belongs to that or that category. This requires that the category names are chosen such that they reflect their common semantics, meaning that (at least most) people will recognize and understand the name of a category and its meaning in the same way.

But how can images be separated and categorized? There are basically two ways to approach the question. The first approach is to look at the main subject of images in general. Some common subjects are people (portraits or group of people), buildings, concerts, landscapes, overviews, animals, churches, flowers, castles, football matches etc. The second approach is to categorize images into categories that will have a significant difference for an automatic image tagging system. For example, images captured at an event must have been taken during a limited time interval, the time interval when the event occurred. However, images taken of objects can be and often are relevant regardless of image capture time. Objects and events are the most distinguishable main categories. However, there exist several different types of objects and events.

5.2.3 Objects

In the InfoAlbum [2] system, objects are divided into sub-categories such as man-made objects, natural objects, landscapes and living things. The categories are used for collecting and ranking relevant information to images from external sources.

However, the question is whether dividing objects this (or any other) way makes any sense for the work in this thesis. For it to make sense, the different types of objects must be handled differently, i.e. the search parameters must be different when searching for relevant images. But there are no such obvious differences between for instance man-made and natural objects.

Landscape and overview images of cities or areas are interesting type of images. These images are both time-based (if season or time of day is of importance) and location-based. Tags most relevant for these images are location names and contextual information such as weather or season information (which as discussed in Chapter 4.4 is not the focus of this work). Therefore, it seems satisfactory to handle these images as objects, and use suitable sub-categories such as *overview*, *city view*, *landscape* or something similar. However, it could be confusing for users to place overview images of cities and landscapes into the category *object*. Therefore, another category named *place* is introduced. This category will be handled the same way as objects and are only made to avoid confusion for the users of the system. Some sub-categories such as *aquarium* and *stadium* can be regarded as *object* or as *place* by different users. However, this is not a problem since *object* and *place* are processed the same way by the system. Thus, *object* and *place* are in essence the same category only with different names.

Images of living things and people (portraits) are a common type of images. The problem is that these images are neither time nor location-based unless one or more objects in the background are significant or the image is taken in connection with some kind of event. If so, they can be divided into one of the two discussed categories *object* or *event* respectively.

To summarize, no noticeable way of dividing objects into more defined categories were found by doing experiments (although place is used as another main category).

5.2.4 Events

Date and time of image capture is stored and available for most digital images. Images taken around the same time as the query image could be relevant for the query image because it is possible that some kind of event have occurred if several images are taken somewhat nearby at the same time.

An event can have short or long duration. While a concert usually last for only some hours, a festival might last for several days or even weeks. Early experiments indicated that using the same method for tagging images from events such as concerts and festivals did not work well. The problem was that using a long time interval on concerts (lasting only for hours) introduced noisy tags, whereas using a short time interval on festivals (lasting for several days) lead to few images being returned. The approach taken is to split into events that last for a short period of time (up to one day) and events that last for a longer period of time (up to around a month).

A second possible way of dividing events, is to take use of that some events are special events only occurring once, whereas some events could occur more than one time based on some kind of pattern in time. For example, an event such as Halloween occurs every year on the exact same date (at least in USA), whereas a specific concert is a special event without any significant time pattern.

For an event such as Halloween it might not be important that images are captured at a specific year as long as the images are from Halloween regardless of which year the images are captured. But for other yearly events such as the Roskilde Festival in Denmark, it is not desirable to tag an image from the festival in 2009 with something related to 2008. Thus, for a given repeating event, it might and it might not be important that images are taken at a certain year as long as they are from the same kind of event but taken in a different year.

Another problem with repeating events is that they can have very different time patterns such as quadrennial (every forth year), yearly, monthly, weekly and so on making them hard to handle. Further, the time patterns can be skewed between each interval (e.g. Easter and Rio Carnival not occurring at exactly same time every year).

While experiments did indicate that it for some scenarios could have been useful to divide and handle one-time events versus repeating events, the gain was small and considered not worth the effort because of the discussed problems. Further, as discussed in 5.2.2, it is not desirable to have too many categories and introducing more event categories would at least double the number of events as there must be support for both short/long-lasting repeating events and short/long-lasting one-time events. There would also be further requirements on user input (e.g. when and how often the events are repeated).

5.2.5 Categorization

To summarize, the four discussed main categories listed below and shown in Figure 5.3 are found to be sufficient for the image tagging system designed in this thesis. Some examples of sub-categories are also shown in Figure 5.3. As previously discussed, sub-categories are user-defined and can be made by users of the system while the main categories are part of the implementation. However, it is possible to add support for more main categories to the implementation later in future work if and when the system is expanded.

1. **Object** – a wide range of objects with sub-categories such as *tower*, *castle, stadium* etc.
2. **Place –** images with sub-categories such as *overview*, *city*, *city view*, *landscape* etc.
3. **Short-lasting event** – events lasting for up to one day with sub-categories such as *concert*, *Halloween*, *football match* etc.
4. **Long-lasting event** – events lasting for up to around a month with sub-categories such as *festival*, *Olympics*, *carnival* etc.

This categorization means that the users of the system will initially only see the sub-categories. However, if their image does not fit into one of the existing sub-categories, they must make a new sub-category. When making a new sub-category, they must choose between one of the four main categories in addition to a name for the new sub-category. The importance of the chosen name is discussed in the evaluation in Chapter 8.4.9.

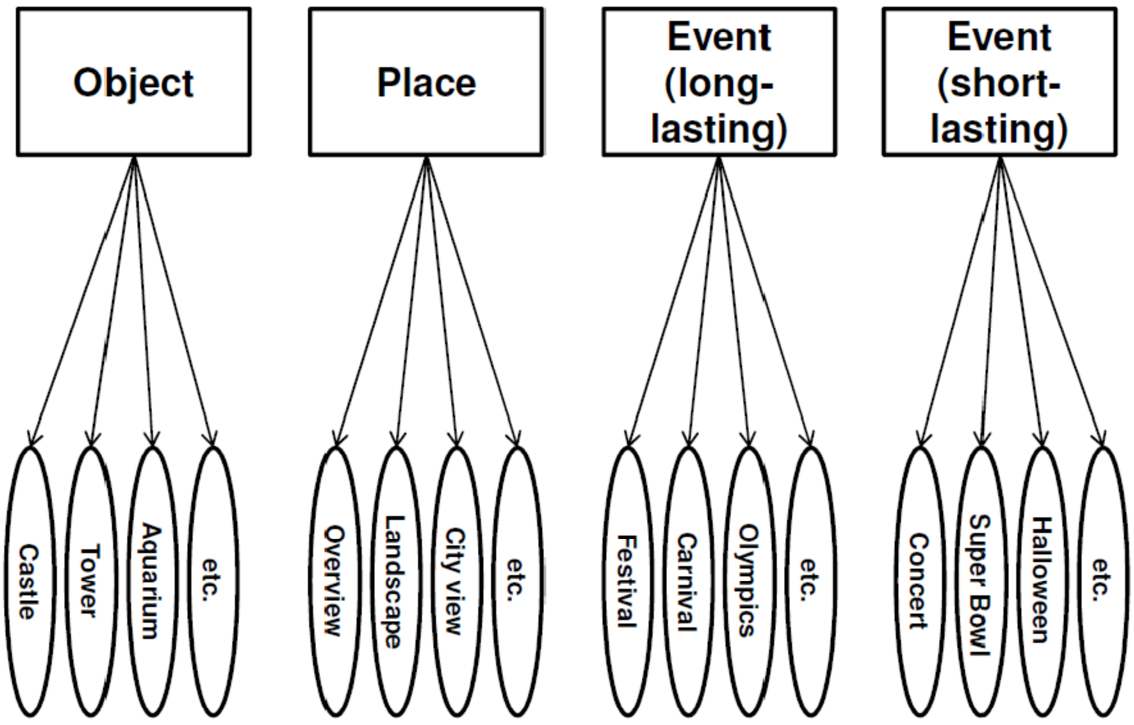


Figure 5.3 – The image categories used in the image tagging system. The sub-categories (bottom) are examples and can be anything that will fit into the main categories (top).

5.2.6 Usage in more detail

The dynamic method used to decide the size of the result set (Chapter 5.1) handles all categories equally. It could have been useful to require more images from objects than for events, but as discussed there are objects with few images available and there are events with many images available. Therefore, a general solution for handling the size of the result set was found to be the better option.

The most important part of the image tagging system is to find a set of relevant images (result set) on Flickr that is to be used as basis for tagging the query image. Because location and location radius are handled equally for all categories, the two changeable search parameters are date/time and sub-category. The usage of the four main categories to specify the search parameters for the image search requests to Flickr is as following:

1. Object
    - Images taken at any date/time can be used (no requirement on date/time of image capture).
    - The sub-category is used to only return images that have the sub-category in its metadata information (i.e. in their title,

description or tags).

2. Place
   - Same as object
3. Short-lasting event
   - Find only images taken inside a time interval of 24 hours before and after the capture time of the query image.
   - Sub-category is not used as search parameter.
4. Long-lasting event
   - Find only images taken inside a time interval of 30 days before and after the capture time of the query image.
   - The sub-category is used to only return images that have the sub-category in its metadata information (i.e. in their title, description or tags).

This means that the search requests to Flickr will vary depending on which category the image belongs to.

The usage of sub-category in the search process of short-lasting events were found to be unnecessary and resulted in usage of even fewer of the available images (there are generally fewer images available for events than for objects). It is not necessary to use the sub-category because almost all images taken during the short time interval are relevant for the query image. Therefore, no further information for preventing the occurrence of images that are not relevant is needed. It could however lead to problems if many short events occur at the same time and place.

The experiments further indicated that the sub-category should be used in the search process for long-lasting events to prevent the occurrence of images that are not relevant. This is because the time interval for long-lasting events is significantly larger than for short-lasting events. Therefore, there might be several events occurring around the same place inside the time interval and thus there is a need to differentiate the events. Further, it is no longer as likely that the images inside the time interval are relevant for the query image since the time interval is larger. Thus, there are more images inside the time interval that is not relevant for the query image, and therefore the sub-category is used as a search parameter for finding relevant images.

5.2.7 Synonyms

The sub-category is used in the image search requests as a search parameter for all the main categories except short-lasting events. This means that only images that have the sub-category (e.g. castle or festival) in its metadata information (title, description or tags) will be used by the system. This could be very restrictive (result in usage of few images) although it is a very good approach for limiting the occurrence of noisy tags. To allow usage of more images (that is relevant), all synonyms of the sub-category are used in the search process, such that images that have the sub-category or a synonym of the sub-category in its metadata information are used. Synonyms are collected using an online API (see Chapter 7.5).

For example, for the sub-category *castle*, the synonyms *palace* and *rook* are returned. The image tagging systems will therefore use all images where one of the words *castle*, *palace* or *rook* appears in the metadata information. This introduce the possibility of more noisy tags appearing because some of the synonyms might not be relevant for the query image (for example, *rook* (piece in the board game of chess) is not likely to be relevant). However, this occurs very seldom and often do not interfere the result (e.g. not many chess images close to a castle). Further, the benefit of making the search process less restrictive (allowing usage of more images that are relevant) is higher (than the possible disadvantage of introducing noise).

5.3 Handling several images from the same user

Some people upload many images with the same set of tags. This can happen if several images are taken from the same event or object. While many of the tags probably are relevant for all the images, it is not desirable to use the same tags from the same user more than once. The reason is non-relevant tags which could for example be the name of the person who captured the image, the name of the (uninteresting) people on the images, the name of the camera used to take the image with, tags that are only relevant for the specific user (e.g. honeymoon) and other similar information. These noisy tags could get a high frequency and therefore be selected as one of the top ranked tags by the automatic image tagging system. Therefore, only one and not more than one image should be allowed from the same user.

However, using only one image per user can lead to a very small result set for query images that have only few relevant images. This is because images from very active users (who captures/uploads many images) are discarded. Currently there are not enough geo-referenced images available to be able to only use one image per user without limiting the number of images available significantly for certain query images. Also, images from active and productive users often have very relevant and good tags compared to images from less active and productive users. To counter for this observation, a method where all images from users that already had an image in the result set were put in a waiting list. The tags of the images in the waiting list were later added as tags if these tags were used by enough of the other users. This lead to significant increase in images in the result set, but it also increased the appearance of noisy tags and made the tag frequencies very uneven. This was because many images from the same user could end up in the result set, and thus tags used by this user (in all of the images from the user) would get an artificial high frequency which made it hard to decide which tags that were relevant.

Further experiments demonstrated that using only one image per user and decreasing the amount of required images in the result set provided best results. A final addition to this approach is made to allow tags from different images by the same user to be used as long as the tags are not similar to tags already added from other images from the same user. Thus, each unique tag can only be used by the image tagging system once per user. In other words, if there already exist an image from Bob in the result set, then a tag from another image from Bob is only used if the tag has not already been added by any other image from Bob that already is in the result set. This allows usage of all unique tags from users who have several relevant images with different sets of relevant tags (initially only tags from one (the first) image per user were used).

5.4 Tag processing

The image tagging system now has a result set of *images* that are regarded to be relevant for the query image (found by using the methods described in Chapter 5.1 – 5.3). The next step is to handle the *tags* of the images in the result set. These tags are to be used to tag the query image.

5.4.1 Deciding how many tags that are relevant

The most obvious approach is to simply tag the query image with the 5 or 10 most frequently used tags among the images in the result set. But there are images where many tags are relevant and there are images where few tags are relevant. It is not desirable to tag a query image with many tags if that only results in adding noisy tags. Contrary, it is not desirable to tag a query image with only x tags if more than x relevant tags are collected. A more dynamic approach is needed.

Another possibility is to discard tags with low frequencies because tags occurring in just some of the images are likely to be noisy, as opposed to tags occurring in many of the returned images. The approach taken is to only use tags that are used in at least 20 % of the images in the result set. Therefore, for a tag to be selected as relevant for the query image, the tag must appear in at least one fifth of the images in the result set. The usage of 20 % was found to be effective for discarding noisy tags.

When there are less than 15 images in the result set, the minimum times a tag must appear in the result set is set to 3 (instead of 20 %) to prevent noisy tags from appearing (for example, 20 % of 10 is 2). The reasoning is that a tag used by 3 different users is not likely to be noisy. Experiments using 2 as the minimum required tag frequency introduced noisy tags, and it is better to not return many tags than to return many tags where several of them are noisy.

Regarding this approach, it is worth mentioning that images without any tags are not used by the system. It is important to discard these images because they would disturb the selection as they would increase the total number of images in the result set without contributing with any tags. This would make it harder for tags in the result set to be selected as relevant. Therefore, images without any tags are not added to the result set. Similarly, images with only one tag are not added either. One reason is that experience shows that the one and only tag from images that only have one tag more often than not is just a noisy tag (often just a whitespace). Another reason is that images with only one tag are not desirable to use as they provide very little valuable information.

5.4.2 Tag filtering

Some tags are not usable; they simply are of no interest to any image. The quite frequently used tag *geotagged* is the most common occurrence of these kinds of tags. Although you could argue that it is of interest to know whether an image is geo-referenced, it is actually of no interest because all the images used by a location-based automatic image tagging system must be geo-referenced. Another example is tags consisting of just a whitespace.

Some people tag images with the camera they have used to capture the image with, and some users (and/or applications) tag images with the name of the application used to edit or upload the image. This is not relevant for the query image, but it could lead to tags with commonly used cameras and/or applications to be selected as tags for the query image.

There is no reason to use these kinds of tags, and they can safely be filtered because they are noisy tags. They are removed with use of a filter list with unwanted tags. The filter list currently consists of tags such as *geotagged*, *flickr*, *(whitespace)*, *latitude* and *nikon*. The tag filtering can be compared to email filtering where spam mail is filtered out.

5.4.3 Handling whitespaces

A whitespace is an empty character in between two words. Some tags such as “Big Ben” consists of more than one word, and the whitespace in the middle is part of the tag. For these kinds of tags, it is also common to tag without whitespaces, which in this case would be “BigBen”. These two tags (“Big Ben” and “BigBen”) can be handled as separate tags by an image tagging system, and this causes two problems. The first problem is that both tags can be assigned as tags to the query image. The second problem is that the combined frequency should be higher, or in other words, the frequency is lower than it should be, because the tag is split up into two separate tags when it is obvious that both tags have the same meaning. This can therefore lead to a situation were neither of the tags are selected (if both have frequencies below 20 % of the number of images in the result set).

In the implementation, this problem is solved with the following steps. First, all duplicate tags are located where removing whitespaces from a tag makes it equal to another tag. Then the less frequent of these two tags is removed, and the frequency of the less frequent tag is added to the more frequent tag. For example, if there are two tags *Big Ben* and *BigBen* with frequencies of 30 and 19 respectively, then the tag *BigBen* is removed, and the tag *Big Ben* will get a new frequency of 30 + 19 = 49.

Upper and lower case characters are handled the same way as whitespaces, meaning that two tags “big ben” and “Big Ben” with frequencies of 4 and 30 respectively would lead to “big ben” being removed and “Big Ben” getting new frequency of 30 + 4 = 34.

However, an optimization of the implementation removed whitespaces and forced all characters into lower case characters. Initially, tags were collected with one request to Flickr for *each* image in the result set resulting (in addition to image search requests) in as many requests to Flickr as images in the result set. The optimization consisted of requesting the tags to be sent along with the *result set* of images returned by the image search request. This way, only the search requests are needed (as tags are now sent along with the result set of images), improving the performance of the system considerably. However, the backside is that Flickr removes any whitespace character and transforms any upper case character into lower case character when the tags of the images are returned along with the result set of images. Whitespaces and upper case characters could be reintroduced at the expense of performance. Another possibility is that Flickr change their API such that whitespaces and upper case characters are not lost when tags are returned in the search request.

5.4.4 Usage of other information than tags

It was tested whether usage of image title and/or image description in addition to image tags would be useful. The short story is that it was not. Most useful words in the title or description are often also included as tags. Further, the image title is often meaningless (e.g. DSC1001), and image descriptions are descriptive sentences and therefore they consist of a lot of regular words. However, this could be handled by known natural language processing and information extraction techniques that for example remove so-called stop words (it, is, the, are etc.) and other irrelevant information. But the usage of these techniques has not been explored as the relevance for this work is limited. It should be enough to use the tags of the images in the result set, and it is not certain that the image descriptions will provide more useful information without introducing noise.

Chapter 6

Design

In this chapter an overall view of the automatic image tagging system is discussed and presented. The name of the system, its user interface, the workflow of the system and the assumptions of the system will be discussed.

6.1 Naming the image tagging system

Until now, the automatic image tagging system designed and implemented in this thesis has been referred to as just that without a proper name. However, it proved useful to give the system a name when describing the design and performing the evaluation. The suggested name is LoCaTagr, which stands for **Lo**cation, **Ca**tegory and **T**ime-based automatic image t**ag**ging system using Flick**r**. The ending (r) indicates that Flickr is used, and therefore also slightly that *images* are tagged since Flickr is a well known *image* sharing database.

6.2 Web interface

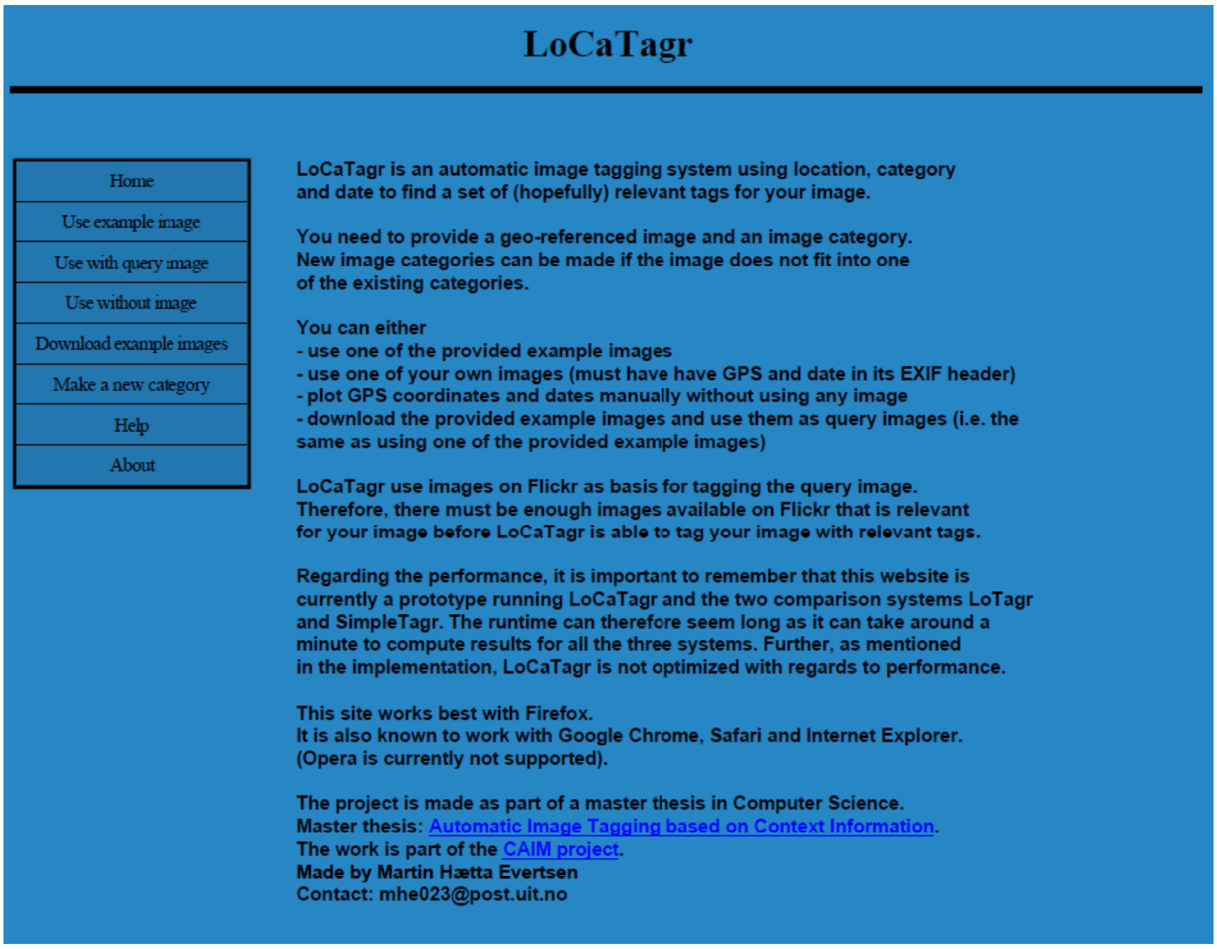
The user interface for testing the implementation is a simple web interface where users can upload .jpg images that have location and date/time information available in the EXIF-header. The category must also be provided by the user. New categories can be made if the image does not fit into one of the existing categories.

Some example images which are used in the evaluation (Chapter 8) can be used as example images for users that do not have .jpg images with location and date/time stored in the EXIF-header of the image available. Further, it is also possible to plot GPS coordinates and date manually without using any image. This is possible because the actual content of the image is not analyzed by LoCaTagr.

The tags selected by the image tagging system are presented in a result table on the web interface. The tags are not stored back to the image because the web interface is only a test environment made to show the results of LoCaTagr rather than being a finished image tagging tool ready for commercialization.

The front page of the web interface is shown in Figure 6.1. The web interface can be accessed on the following address:

- http://caim00.cs.uit.no/LoCaTagr



LoCaTagr is an automatic image tagging system using location, category and date to find a set of (hopefully) relevant tags for your image.

You need to provide a geo-referenced image and an image category.
New image categories can be made if the image does not fit into one of the existing categories.

You can either
- use one of the provided example images
- use one of your own images (must have have GPS and date in its EXIF header)
- plot GPS coordinates and dates manually without using any image
- download the provided example images and use them as query images (i.e. the same as using one of the provided example images)

LoCaTagr use images on Flickr as basis for tagging the query image.
Therefore, there must be enough images available on Flickr that is relevant for your image before LoCaTagr is able to tag your image with relevant tags.

Regarding the performance, it is important to remember that this website is currently a prototype running LoCaTagr and the two comparison systems LoTagr and SimpleTagr. The runtime can therefore seem long as it can take around a minute to compute results for all the three systems. Further, as mentioned in the implementation, LoCaTagr is not optimized with regards to performance.

This site works best with Firefox.
It is also known to work with Google Chrome, Safari and Internet Explorer.
(Opera is currently not supported).

The project is made as part of a master thesis in Computer Science.
Master thesis: Automatic Image Tagging based on Context Information.
The work is part of the CAIM project.
Made by Martin Hætta Evertsen
Contact: mhe023@post.uit.no

Figure 6.1 – Front page of the web interface. The simple menu is located to the left.

6.3 Flickr

Flickr offers a public API which can be used to write applications that use Flickr in some way. LoCaTagr use Flickr to find a set of relevant images and retrieve tags from these relevant images.

Since Flickr is currently the only information source used, LoCaTagr is limited in that it will only work using Flickr and the images that are available there. Supporting more image sharing applications would increase the number of images but the type of images in these other image sharing applications are likely to be similar to what can be found on Flickr. Therefore, the limitation becomes which images that are typically uploaded to image sharing applications.

6.4 Overall view

The overall workflow in LoCaTagr (and its web interface) is listed below. A graphical overall view is shown in Figure 6.2 where LoCaTagr is represented by the blue box on the right hand side of the image.

1. The user provides a query image and image category through a simple web interface that start the execution of the LoCaTagr system.
2. LoCaTagr reads GPS coordinates and date/time from the EXIF-header of the provided query image.
3. LoCaTagr generates a search based on which category the query image belongs to (discussed in Chapter 5.2).
4. The search query is sent to Flickr (initially with a very small radius).
5. Flickr returns a set of images.
6. LoCaTagr adds images that are considered to be relevant for the query image to the result set (discussed in Chapter 5).
7. As long as not enough required images are added to the result set, LoCaTagr decreases the number of required images in the result set and performs a new search (step 4) using a radius double as big as in last search (discussed in Chapter 5.1 and Table 5.2)
8. LoCaTagr performs tag processing (discussed in Chapter 5.4) when the result set has reached required amount of images.
9. The result (list of tags for the query image) is presented to the user through the web interface (see next sub-chapter, Chapter 6.5).

6.5 Presenting the result

The tags found by LoCaTagr are presented in a table on the web interface. The frequencies of the tags are also shown in the table. Below the list of tags found by LoCaTagr is a comparison table showing the result of LoCaTagr together with the result from two other comparison systems. The result page for Image 4.1 is shown in Figure 6.3.

The two other comparison systems do not use category and date/time. The first (LoTagr) is actually LoCaTagr without the usage of category and date/time. The other system (SimpleTagr) is a very simple image tagging sy-

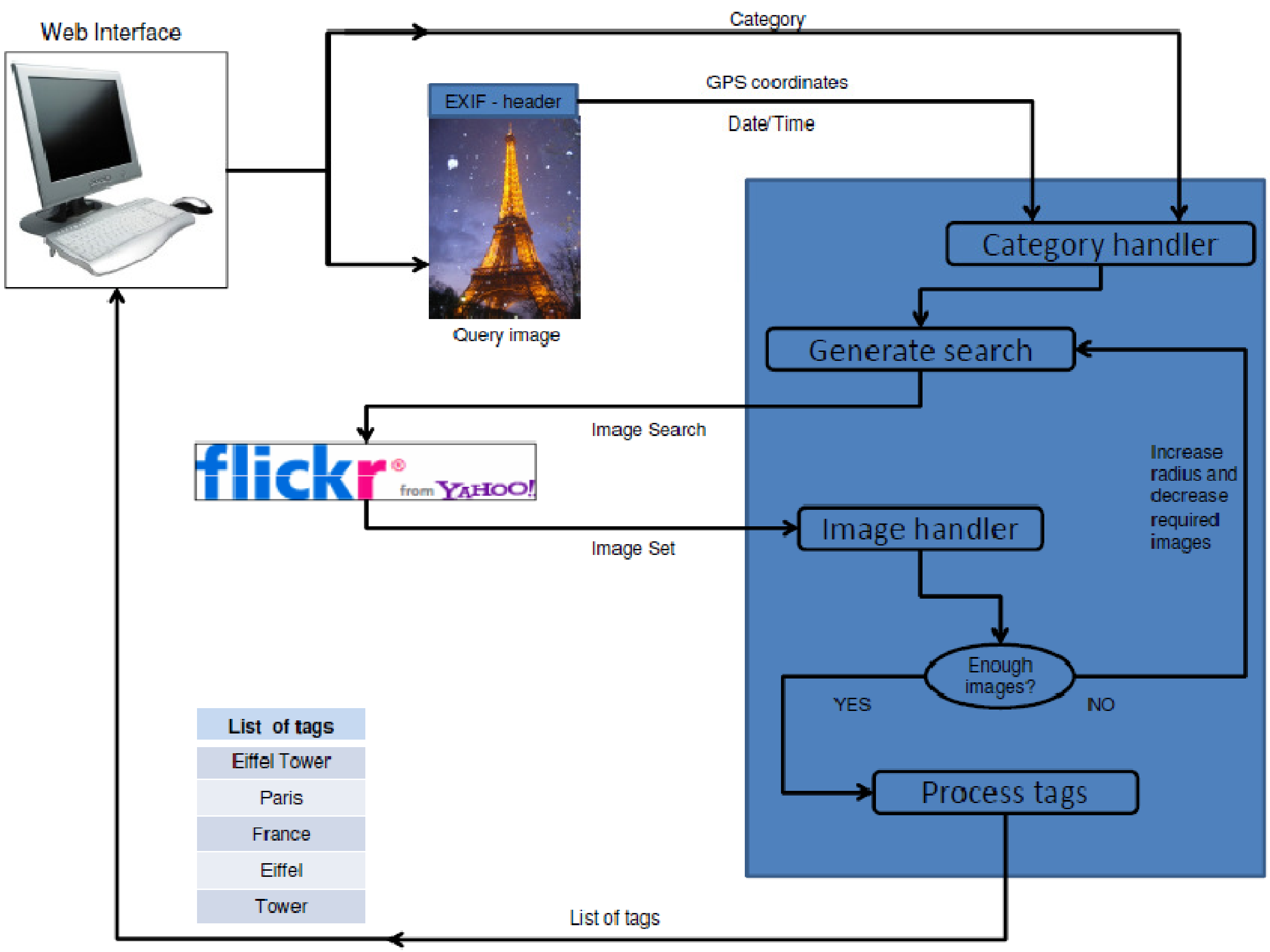
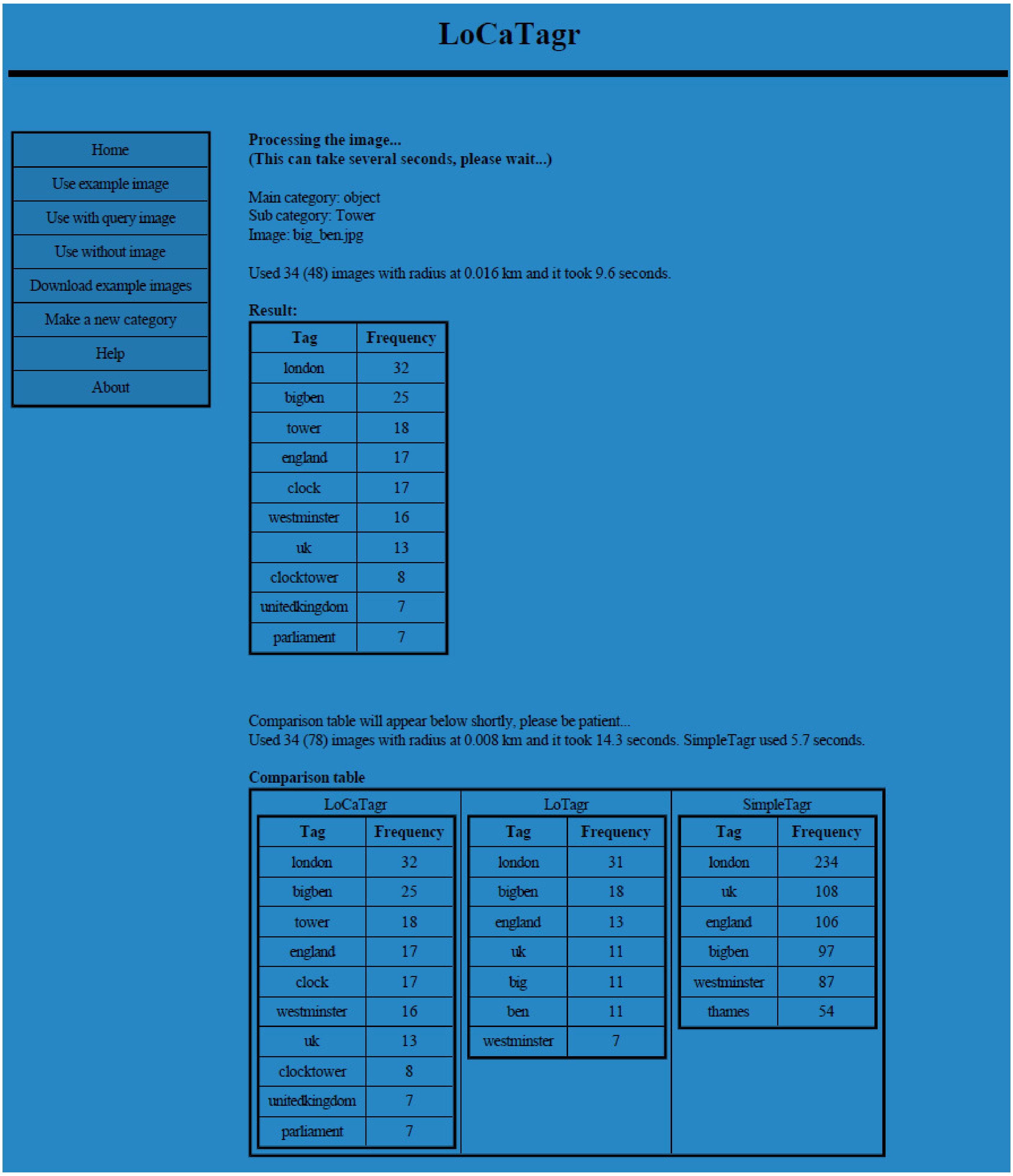


Figure 6.2 – A graphical overview of LoCaTagr (blue box on right hand side) and the workflow between the system, web interface and Flickr.

stem using only location. Thus, the first system (LoTagr) use the optimizations discussed in Chapter 5 (restricting usage of images from same user, tag filtering etc.) whereas the other (SimpleTagr) does not. Because these two systems use only location, all images are handled the same way regardless of categories and date/time of image capture. Comparison against these systems is useful for evaluating whether usage of image categories is useful. The two comparison systems will be discussed in more detail in Chapter 8.1.

6.6 Assumptions

It is assumed that there exist a representative set of already geo-referenced and tagged images on online image sharing databases such as Flickr. It is not possible to find relevant tags for a query image if there are not enough a-



Processing the image...
(This can take several seconds, please wait...)

Main category: object
Sub category: Tower
Image: big_ben.jpg

Used 34 (48) images with radius at 0.016 km and it took 9.6 seconds.

Result:

Tag	Frequency
london	32
bigben	25
tower	18
england	17
clock	17
westminster	16
uk	13
clocktower	8
unitedkingdom	7
parliament	7

Comparison table will appear below shortly, please be patient...
Used 34 (78) images with radius at 0.008 km and it took 14.3 seconds. SimpleTagr used 5.7 seconds.

Comparison table

LoCaTagr		LoTagr		SimpleTagr	
Tag	**Frequency**	**Tag**	**Frequency**	**Tag**	**Frequency**
london	32	london	31	london	234
bigben	25	bigben	18	uk	108
tower	18	england	13	england	106
england	17	uk	11	bigben	97
clock	17	big	11	westminster	87
westminster	16	ben	11	thames	54
uk	13	westminster	7		
clocktower	8				
unitedkingdom	7				
parliament	7				

Figure 6.3 – The result page for Image 4.1 using tower as the sub-category. The table at the middle of the page is the result from LoCaTagr. The comparison table at the bottom shows the result from LoCaTagr to the left, LoTagr in the middle and SimpleTagr to the right.

vailable images on Flickr that are relevant for the query image. However, LoCaTagr should work for the most common and interesting type of images because these images are likely to have a representative set of relevant images available. Furthermore, more and more images will be supported as the number of geo-referenced images in online image sharing databases continues to grow at high speed.

It is assumed that the user specifies an image category along with the query image that is to be tagged. We think that users will take the effort of providing image categories if that can provide useful and valuable information to the images. As long as the user interface for the categorization process is rather structured and simple, the effort in choosing categories should be affordable. It is also possible that this assumption could be avoided in future work if a method for automatic categorization is possible to implement (discussed in Chapter 9). It is the users who will choose or create the subcategories, and that put some requirements on the user. Choosing wrong or badly named categories could lead to finding tags that are not relevant, or that the system does not find any tags (discussed in Chapter 8.4.9).

It is also assumed that the location where the image was captured is available in the form of GPS coordinates (latitude and longitude) in the EXIF-record of the query image. This assumption could be avoided if allowing the user the possibility to manually geo-reference the image through a map interface. Further, it is likely that many cameras and mobile devices in the future will be equipped with GPS systems and thus store capture location in the EXIF-header of the images.

Finally, it is assumed that the date and time the image was captured is available. This assumption is almost redundant because practically all digital images have time of image capture available. Further it is assumed that users set the clock on their digital camera correct. However, small errors in date/time (up to some hours for short-lasting events and up to some days for long-lasting events) will not make any significant difference because the time intervals used in the system are sufficiently large to cover for such small errors. Time is only a requirement for event images.

Chapter 7

Implementation

In this chapter some implementation specific details are presented.

7.1 Hardware

LoCaTagr is implemented and tested on an Intel® Core™ 2 Duo CPU P9700 2.80 GHz with 4 GB RAM running Microsoft® Windows XP Professional SP3.

The server hosting the web interface and LoCaTagr is an Intel® Core™ 2 CPU 6400 2.13 GHz with 2 GB RAM. The system is running Microsoft® Windows Server 2008 R2 Standard v6.1.

7.2 LoCaTagr

LoCaTagr is written in Python (v2.6.5), and the code can be found in Appendix B. The performance of the system has not been prioritized as the main objective is to investigate whether usage of categories together with location and date/time is useful. For example, the list with tags is iterated several times for different operations like tag filtering, white space handling and similar operations when it would have been more efficient to iterate the list once. Further, a high-level programming language (that generally has slower performance than low-level programming languages) is used.

The runtime of LoCaTagr is around 22 seconds on average. This will be more discussed in Chapter 8.4.11. LoCaTagr is fastest when many relevant images from different users are found close to the query image. The run-time varies for each image mainly because the number of relevant images on Flickr varies for each query image. The run-time (and result) can also vary for the same image because relevant images can be deleted, added or be temporarily unavailable. Other reasons for run-time variations can be bandwidth fluctuations, workload on Flickr and workload on LoCaTagr.

The process taking most time is sending image requests to Flickr and retrieving image sets from Flickr. As indicated in Table 5.2, a maximum of 16 different search radiuses is used. More than one search request for each search radius can be needed if more than 250 images (which is the maximum number of images returned by Flickr for one request) are returned in the result sets. This can occur if there are many images from the same users in the result set (as LoCaTagr only use one image per user as discussed in Chapter 5.3) or if many images do not have any tags (as these images are not used by LoCaTagr as discussed in Chapter 5.4.1). However, most query images require only between 1 to 7 search requests depending on how many relevant images that are available.

7.3 Web interface

The simple web interface is written using Perl (v5.10.1) scripts, and is intended as a test environment for the implementation. Error checking and response messages are thus limited to a basic level. Further, the graphical design of the web interface has not been prioritized. The code for the web interface can be found in Appendix C.

Apache HTTP server is used to host the web interface. The Apache HTTP Server Project[23] is an effort to develop and maintain an open-source HTTP server for modern operating systems where the goal is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

Mozilla Firefox 3.6.3 is used to test the web interface. The web interface is also known to work with Google Chrome, Safari and Internet Explorer. However, Opera is not supported.

7.4 Flickr API

Flickr offers a public API available for non-commercial use by outside developers that makes it possible to write applications that use Flickr some way or another. LoCaTagr use Flickr to find a set of relevant images and retrieve tags from these relevant images. An API key and an API secret is required for usage of the Flickr API.

[23] http://httpd.apache.org/

A Python FlickrAPI[24] (v1.4.2) is used for handling connection (with the API key and API secret) and search requests to Flickr. The Python Flickr API claims to be the easiest to use, most mature and feature-rich Python interface to Flickr**.**

7.5 Synonyms

Synonyms are collected using the STANDS$_4$ Web Services - Synonyms API v1[25]. This system was preferred over WordNet, mainly because it was easier to use as WordNet is rather advanced (but the advanced features are not needed by LoCaTagr).

As discussed in Chapter 5.2, the sub-category of the image is used in the search process (except for short-lasting events), such that only images where the sub-category appears in the title, description or tags of the image are used. This could be restrictive and lead to usage of few images. Therefore, the synonyms of the sub-category are collected and used in the search process so that only one of the synonyms of the sub-category must appear in the metadata of the image for it to be used by LoCaTagr (discussed in Chapter 5.2.7).

7.6 EXIF-header

LoCaTagr use a Python library (Gene Cash's EXIF.py library 1.1.1[26]) to extract EXIF information from digital image files. GPS coordinates (latitude and longitude) and date/time information is collected from a .jpg image. The GPS coordinates are transformed to decimal degree form as it is the most convenient way to represent GPS coordinates in a computer system.

Manual manipulation of EXIF-headers (adding GPS coordinates to image files) was done with a tool called ExifTool (v8.19). ExifTool[27] is a platform-independent Perl library and command-line application for reading, writing and editing metadata information in a wide variety of files (including .jpg files). ExifTool supports many different metadata formats including EXIF and GPS. Manually geo-coding images were more convenient than searching for already geo-referenced images that were usable (metadata information is lost when downloading images from Flickr).

[24] http://stuvel.eu/projects/flickrapi
[25] http://www.abbreviations.com/synonyms_api.asp
[26] http://sourceforge.net/projects/exif-py/
[27] http://www.sno.phy.queensu.ca/~phil/exiftool/

Chapter 8

Results and Evaluation

In this chapter the results from LoCaTagr and some comparison systems are presented and evaluated.

8.1 Comparison systems

LoCaTagr will be compared and evaluated against the following systems

1. LoTagr
   - LoCaTagr using only location (not category and date/time information). As this system does not use category, the "Ca" part of the name is removed and this system will be referred to as LoTagr in the comparison. Thus, LoTagr is the LoCaTagr system using location and the tag and image processing techniques discussed in Chapter 5 (e.g. restricting usage of images from same users and tag filtering) except those related to category and date/time.

2. SimpleTagr
   - A simple location-based image tagging system. It uses only location (not category and date/time). It will simply use the up to 250 images that are closest to the query image and return all tags that appear in at least 20 % of the images found. Other than this dynamic approach for deciding how many tags that are relevant, SimpleTagr do not use the image and tag processing techniques (e.g. restricting usage of images from same users and tag filtering) used by LoCaTagr and LoTagr.

3. SpiritTagger v0.3 [28]

[28] http://cortina.ece.ucsb.edu/index.php

- SpiritTagger is presented in Chapter 3.1 and is an image tagging suggestion tool based on location and content-based image analysis. It does not use category and date/time. SpiritTagger use Flickr to find relevant tags. An important thing to notice is that the purpose of SpiritTagger is to suggest a set of tags to the user, meaning that the user is supposed to manually select the relevant tags from a list of suggested tags. Therefore, SpiritTagger will always suggest exactly 20 tags for each query image.

SpiritTagger is chosen as a comparison system because it is interesting to compare against SpiritTagger since it use Flickr and location (just as LoCaTagr) in addition to content-based image analysis (whereas LoCaTagr use category and date/time instead). This makes it possible to compare usage of content-based image analysis with the category approach. Further, SpiritTagger has a demo available online, whereas most of the other systems discussed in the related work in Chapter 3 do not have a demo available online.

8.2 Comparison method

The decision whether a tag is relevant or not is done manually by the author of this thesis. It is not possible to ask an automated system for assistance as the decisions are subjective. Further, it was not time or resources available to do a user test with external users.

The following color codes are used in the comparison tables:
- Green tags are regarded as relevant for the query image.
- Purple tags are difficult to decide, and will be referred to as unsure tags.
- Red tags are noisy tags (not relevant for the query image).
- Blue tags are place names.

Recall (number of relevant tags found / number of relevant tags that could have been found) is not used in the evaluation as it near impossible to decide how many tags that could be used to tag an image. It would possibly be even more correct to use total number of relevant tags that are available on Flickr for a certain image. But again, it is near impossible to find all relevant tags that could and should be used for a certain image.

Precision (number of relevant tags divided by total number of tags found) is interesting. A high precision score will prove that most of the tags found are relevant. A perfect precision score (1.00) means that all tags found are relevant.

However, a problem with precision scores is the place names. As previously discussed (Chapter 4.4), place names (with accuracy ranging from continent to street name) can be retrieved by sending the GPS coordinates to external sources such as GeoNames. The problem is to decide the accuracy of the place names and how many of the place names that should be used. Therefore, all place names are disregarded in the calculation of precision.

Another issue is the so-called unsure tags. Often, some will regard them as relevant whereas others will not. Typical tags of this type are tags relevant for the exact position of the image (but not visible on the image). Another example is whether the two tags *big* and *ben* are relevant on their own (not combined) for an image of Big Ben. Therefore, two different precision scores are calculated for each image, where the first (Precision1) is stricter than the second (Precision2):

- Precision1 = relevant tags / (total tags – place names)
- Precision2 = (relevant tags + unsure tags) / (total tags – place names)

Thus, Precison1 will regard the unsure tags as noisy (not relevant), and Precision2 will regard unsure tags as relevant. Total tags are the total number of tags found or suggested for an image.

SpiritTagger will always suggest exactly 20 tags whereas the other systems have a more dynamic approach which results in selecting tags only when they have a certain frequency, i.e. when they appear in at least 20 % of the images in the result set. This is a big disadvantage for SpiritTagger with regards to the precision scores. The idea in SpiritTagger is that the system suggests a set of tags and that the user is supposed to pick the most relevant. It is therefore semi-automatic whereas LoCaTagr is fully automatic (except that an image category is required from the user in LoCaTagr).

Therefore, the number of relevant tags found by each of the systems is also considered. Further, it could be regarded as better to find 10 relevant tags with a quite good precision score (e.g. 0.80 – 0.90) than finding only 1 or 2 relevant tags with perfect precision score (1.00).

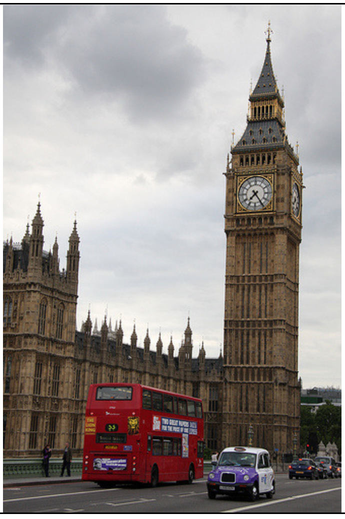
8.3 Images and results

The images used in the evaluation are collected from Flickr. The source where the images are found is listed in Appendix A.

The results will be presented in the next 11 pages. The filename below the image number is the filename of the image as it appears on the web interface. These example images can be used directly as query images on the web interface.

The main and sub-category is used by LoCaTagr but not the other three comparison systems. The short description will point out important information regarding the query image. The ordering of the tags is based on the frequency, i.e. the higher the tag appears in the list, the more frequent the tag was (note that this might not apply to the results from SpiritTagger).

The results will be evaluated and discussed in the next sub-chapter (Chapter 8.4). A table with number of tags found by each system for each image and a table with the average number of tags found by each system can be found in Table 8.7 and Table 8.8 respectively in the summary of the evaluation in Chapter 8.4.13. Summaries of the results are also shown graphically in Figure 8.5, 8.6, 8.9, 8.10 and 8.11 towards the end of Chapter 8.

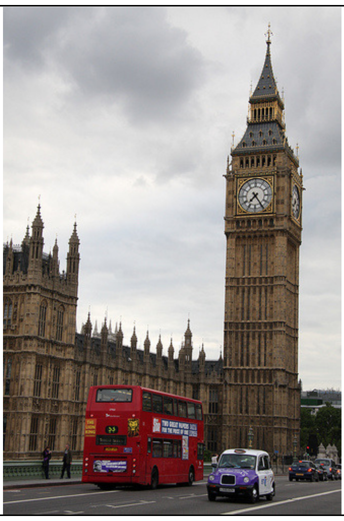
Image 1A big_ben.jpg Main category: object Sub-category: tower Big Ben in London. Geo-referenced at exact location of Big Ben which is part of Houses of Parliament.	

	LoCaTagr	**LoTagr**	**SimpleTagr**	**SpiritTagger**
	london	london	london	London
	bigben	bigben	uk	England
	tower	england	england	United Kingdom
	england	uk	bigben	UK
	clock	big	westminster	parliament
	westminster	ben	thames	City
	uk	westminster		big
	clocktower			ben
	unitedkingdom			clock
	parliament			europe
	big			tower
	ben			river
				night
				Great Britain
				clock tower
				underground
				tube
				Britain
				travel
				cycling
Statistics				
Total	12	7	6	20
Relevant	5	1	1	5
Place N.	5	4	4	7
Unsure	2	2	1	3
Noisy	0	0	0	5
Precision1	0,71	0,33	0,50	0,38
Precision2	1,00	1,00	1,00	0,62

Image 1B
big_ben2.jpg

Main category: object
Sub-category: tower

Geo-referenced at Westminster Bridge on River Thames (location of image capture).



LoCaTagr	LoTagr	SimpleTagr	SpiritTagger
London	london	london	London
England	england	uk	England
bigben	uk	england	United Kingdom
Westminster	londoneye	bigben	UK
uk	thames	thames	parliament
tower	river	westminster	river
clock	westminsterbridge	londoneye	City
parliament	westminster	river	europe
clocktower	bigben		big
unitedkingdom	unitedkingdom		ben
westminsterbridge			clock
thames			Houses
housesofparliament			tower
riverthames			bridge
night			night
			Britain
			Great Britain
			palace
			vacation
			architecture

Statistics				
Total	15	10	8	20
Relevant	6	1	1	6
Place N.	5	5	4	7
Unsure	3	3	2	5
Noisy	1	1	1	2
Precision1	0,60	0,20	0,25	0,46
Precision2	0,90	0,80	0,75	0,85

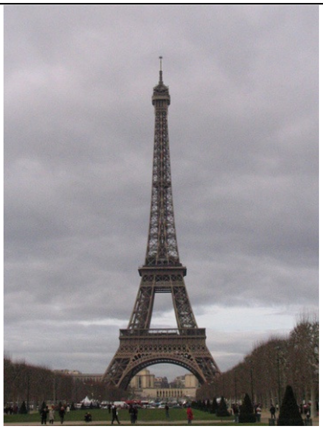
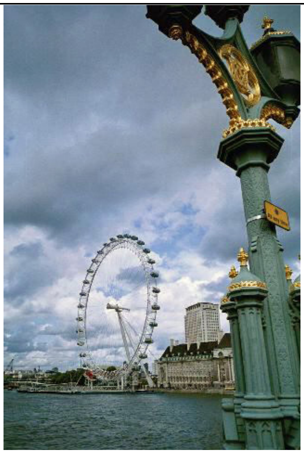
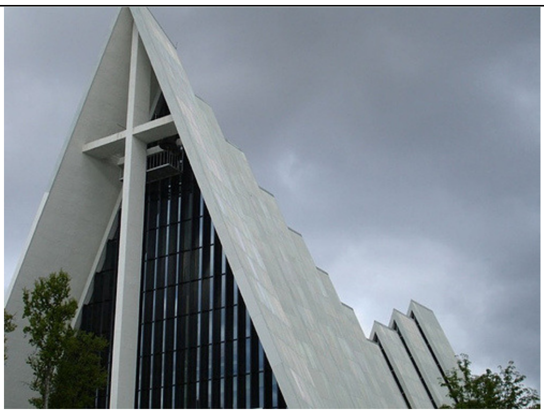
	Image 2 eiffel_tower.jpg Main category: object Sub-category: tower Eiffel Tower in Paris. Geo-referenced at exact position of the tower.			
	LoCaTagr	**LoTagr**	**SimpleTagr**	**SpiritTagger**
	paris	paris	paris	France
	france	france	france	tower
	eiffeltower	eiffeltower	eiffeltower	Europe
	tower	eiffel	eiffel	tour
	eiffel	tower	toureiffel	night
			tower	torre
			Europe	blue
			tour	Bleu
			night	nuit
				CANON
				Lights
				holiday
				City
				architecture
				view
				steel
				monument
				travel
				eu
				sky
Statistics				
Total	5	5	9	20
Relevant	3	3	3	6
Place N.	2	2	3	3
Unsure	0	0	0	3
Noisy	0	0	3	8
Precision1	1,00	1,00	0,50	0,35
Precision2	1,00	1,00	0,50	0,53

Image 3 london_eye.jpg Main category: object Sub-category: Ferris wheel London Eye (a Ferris wheel). The image is taken from Westminster Bridge on River Thames in London.	

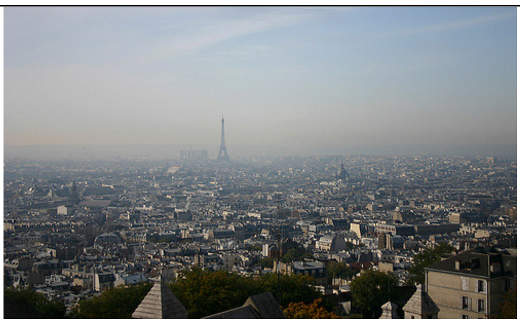
	LoCaTagr	LoTagr	SimpleTagr	SpiritTagger
	london	london	london	London
	londoneye	england	uk	England
	ferriswheel	uk	england	United Kingdom
	thames	londoneye	bigben	UK
	river	thames	thames	parliament
	england	river	westminster	river
	uk	westminsterbridge	londoneye	City
	unitedkingdom	westminster	river	big
	wheel	bigben		ben
	riverthames	unitedkingdom		clock
	europe			europe
	britain			Houses
	greatbritain			tower
	southbank			bridge
	millenniumwheel			night
	eye			britain
				Great Britain
				palace
				sky
				architecture
Statistics				
Total	16	10	8	20
Relevant	7	3	3	4
Place N.	8	5	4	7
Unsure	1	1	0	1
Noisy	0	1	1	8
Precision1	0,88	0,60	0,75	0,31
Precision2	1,00	0,80	0,75	0,38

Image 4 ishavskatedralen.jpg Main category: object Sub-category: church The church Ishavskatedralen (Arctic Cathedral) in Tromsø, Norway.	

	LoCaTagr	**LoTagr**	**SimpleTagr**	**SpiritTagger**
	Norway	norway	norway	Troms Fylke
	church	troms	troms	Norway
	troms	tromso	norwegen	mountain
	cathedral	church	geotagged	Sweden
	tromso	norge		Snow
	architecture	cathedral		norge
	window	architecture		sea
	snow	norwegen		ice
	norwegen	eismeerkathedrale		sunset
	kirche			travel
	ishavskatedralen			road
	eismeerkathedrale			arctic
	arctic			architecture
	architektur			northern
				lamflickr
				campint
				tent
				girlfriend
				Design
				cold
Statistics				
Total	14	9	4	20
Relevant	8	4	0	2
Place N.	4	5	3	3
Unsure	1	0	0	3
Noisy	1	0	1	12
Precision1	0,80	1,00	0,00	0,12
Precision2	0,90	1,00	0,00	0,29

Image 5 cineaqua.jpg Main category: object Sub-category: aquarium Image from an aquarium named Cineaqua close to the Eiffel Tower in Paris.	

	LoCaTagr	**LoTagr**	**SimpleTagr**	**SpiritTagger**
	paris	paris	paris	France
	france	france	france	night
	aquarium	eiffeltower	eiffel	europe
	trocadero	trocadero	eiffeltower	tour
	cineaqua	toureiffel	toureiffel	Bleu
	parigi	eiffel	tower	blue
	francia	europe	tour	nuit
	fish		europe	City
	Europe		(whitespace)	tower
	blue		trocadero	Ciel
	bleu		night	Nikon
				torre
				Lights
				monument
				vacation
				holiday
				capital
				light
				sky
				smoke
Statistics				
Total	11	7	11	20
Relevant	5	0	0	2
Place N.	6	4	4	2
Unsure	0	0	0	3
Noisy	0	3	7	13
Precision1	1,00	0,00	0,00	0,11
Precision2	1,00	0,00	0,00	0,28

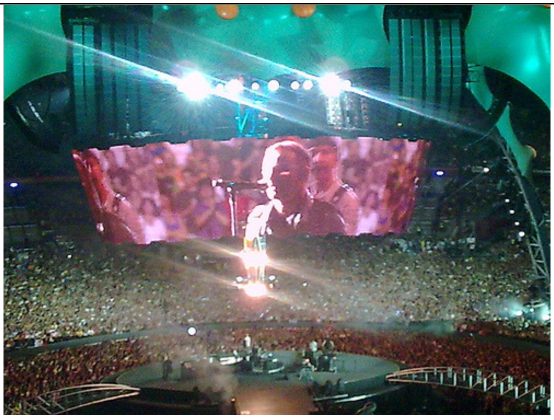
Image 6 paris_overview.jpg Main category: place Sub-category: overview Overview image of Paris taken from the Sacre Cour church which is located on a hill (tertre) named Montmartre.	

	LoCaTagr	**LoTagr**	**SimpleTagr**	**SpiritTagger**
	paris	paris	paris	France
	france	montmartre	montmartre	church
	overview	france	france	HDR
	city	placedutertre	sacrecouer	Europe
	view	tertre		sky
	panorama			holiday
	Europe			blue
	notredame			street
				view
				clouds
				Photomatix
				Eglise
				skyline
				red
				cathedral
				Corazon
				Iglesia
				white
				Pentax
				shop
Statistics				
Total	8	5	4	20
Relevant	3	0	0	6
Place N.	3	4	3	2
Unsure	1	1	1	5
Noisy	1	0	0	7
Precision1	0,60	0,00	0,00	0,33
Precision2	0,80	1,00	1,00	0,61

Image 7
u2_camp_nou.jpg

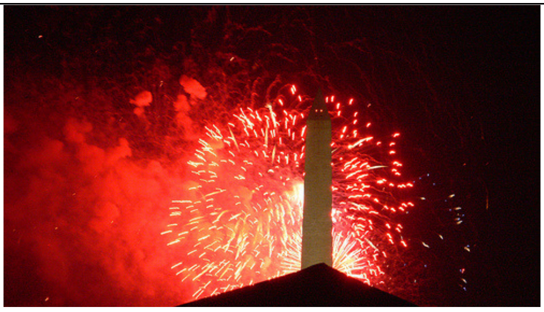
Main category: short-lasting event
Sub-category: concert

U2 concert at Camp Nou, Barcelona during their 360 Tour the summer of 2009.



LoCaTagr	LoTagr	SimpleTagr	SpiritTagger
barcelona	barcelona	barcelona	Do not suggest any tags for this image. The reason is not known.
u2	barca	campnou	
campnou	spain	catalunya	
360	campnou	football	
spain	nou	fcbarcelona	
concert	futbol	spain	
bono	camp	futbol	
tour	football	catalonia	
noucamp	soccer	barca	
edge	fcbarcelona	espana	
concierto		soccer	
theclaw		spanien	
catalonia		europa	
catalunya		europe	
		noucamp	
		stadium	
		cataluna	

Statistics				
Total	14	10	17	0
Relevant	10	1	3	0
Place N.	4	2	9	0
Unsure	0	2	0	0
Noisy	0	5	5	0
Precision1	1,00	0,13	0,38	n/a
Precision2	1,00	0,38	0,38	n/a

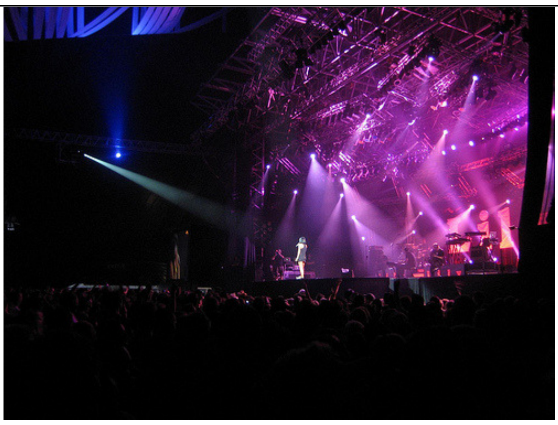
Image 8 independence_day.jpg Main category: short-lasting event Sub-category: Independence Day Fireworks behind the Washington Monument on July 4th (Independence Day).	

	LoCaTagr	**LoTagr**	**SimpleTagr**	**SpiritTagger**
	washingtondc	washingtondc	washingtondc	Washington
	dc	washington	(whitespace)	United States
	washington-monument	washington-monument	washington-monument	District of Columbia
	washington	dc	washington	memorial
	fireworks	usa	dcist	monument
	4thofjuly	crowd	dc	WWII
		obama		Lincoln
		monument		war
		dcist		world
				II
				water
				fountain
				use
				World War II
				night
				pool
				sky
				history
				Independence Day
				nikon
Statistics				
Total	6	9	6	20
Relevant	3	2	1	3
Place N.	3	4	3	3
Unsure	0	0	0	3
Noisy	0	3	2	11
Precision1	1,00	0,40	0,33	0,18
Precision2	1,00	0,40	0,33	0,35

Image 9
roskilde_2009.jpg

Main category: long-lasting event
Sub-category: festival

Roskilde Festival in Denmark the summer of 2009. Orange is a stage, but not the stage on the image.



LoCaTagr	LoTagr	SimpleTagr		SpiritTagger
festival	roskildefestival	festival		Roskilde
roskildefestival	roskilde	roskilde		Sjaelland
roskilde	festival	roskildefestival		Denmark
rf09	denmark	d(e/a)n(e)mark		water
2009	orange	lastfm:event=45113		architecture
denmark	music	rockphotography		statue
orange	concert	photography		church
crowd	roskildefestival2006	musicphotography		summer
roskildefestival2009	rf09	rockmusik		Europe
night	orangestage	metal	low	boat
music	light	hard	heavy	City
	crowd	festivals	rock	blue
		d100	music	sky
		2006	live	Scandinavia
		alternative	nikon	fountain
		progressive	guitar	LIBRARY
		light	bass	white
		tool	lowlight	Bike
		summer	sjaelland	iPhone
		drums	available	building

Statistics				
Total	11	12	33	20
Relevant	8	7	7	1
Place N.	2	2	5	5
Unsure	0	0	6	3
Noisy	1	3	15	11
Precision1	0,89	0,70	0,25	0,07
Precision2	0,89	0,70	0,46	0,27

Image 10 olympics_2010.jpg Main category: long-lasting event Sub-category: Olympics Winter Olympics 2010 in BC Place Stadium in Vancouver, Canada.	

LoCaTagr	**LoTagr**	**SimpleTagr**	**SpiritTagger**
vancouver	vancouver	vancouver	Vancouver
olympics	canada	canada	British Columbia
canada	bcplace	bc	Canada
bcplace	britishcolumbia	britishcolumbia	iPhone
vancouver2010	football	downtown	downtown
2010	bc		WeatherBug
winterolympics	2010		wall
winter	vancouver2010		food
bc	olympics		City
van2010	downtown		Rain
olympic	cfl		Chinatown
britishcolumbia	bclions		building
games			street
			window
			alley
			Mostly Sunny
			Chinese
			roof
			home
			cloud

Statistics				
Total	13	12	5	20
Relevant	9	4	0	0
Place N.	4	4	4	4
Unsure	0	1	0	4
Noisy	0	3	1	12
Precision1	1,00	0,50	0,00	0,00
Precision2	1,00	0,63	1,00	0,25

8.4 Evaluation

LoCaTagr will now be evaluated and compared against the comparison systems. The evaluation will be divided into sub-chapters of the following type of images:

1. Famous attractions (Image 1A and Image 2)
2. Attractions taken from distance (Image 1B and Image 3)
3. Not so famous attractions (Image 4)
4. Not so famous attractions near famous attraction (Image 5)
5. Overview images (Image 6)
6. Short-lasting events (Image 7 and Image 8)
7. Long-lasting events (Image 9 and Image 10)

The discussion of each of the scenarios will have the following general structure: the first paragraph will describe the type of images, the second paragraph will look at the results from LoCaTagr, the third paragraph will compare to LoTagr and SimpleTagr, and the last paragraph will compare against SpiritTagger. Tags will be written in *italics*.

8.4.1 Famous attraction

Image 1A and Image 2 fall into this type of images that are taken of famous and well known attractions where a lot relevant images can be found. Further, these images are geo-referenced at the exact position of the attractions.

LoCaTagr finds a couple of relevant tags (5 and 3) whereas no noisy tags are found. However, *big* and *ben* can be discussed and are therefore regarded as unsure. The problem is that they do not give much meaning on their own (without being combined) but it would be harsh to regard them as noisy. The precision scores are perfect (1.00) or 0.71 if *big* and *ben* is regarded as noisy.

LoTagr and SimpleTagr only found one relevant tag for Image 1A (whereas LoCaTagr found 5). For Image 2, LoTagr returns the exact same set of tags as LoCaTagr whereas SimpleTagr find the same set of relevant tags as LoCaTagr, but also 3 noisy tags. The quite good performance by these two systems indicate that tagging these kinds of images are quite easy, but that using categories does assist and give both more relevant tags and fewer noisy tags.

Interestingly, SpiritTagger suggest neither *Big Ben* nor *Eiffel Tower*, which one would think should be easy to recognize visually. However, it does seem to indicate that the attractions have been recognized by suggesting *big* and *ben* for Image 1A and *tower*, *steel*, and *monument* for Image 2. SpiritTagger is able to suggest a set of interesting tags for Image 2 (*monument*, *architecture*, *steel, City* and *sky*) that are probably recognized by visual similarity techniques as they are not found by LoCaTagr. It therefore finds more relevant tags than LoCaTagr for Image 2, whereas the two systems find almost the same set of relevant tags for Image 1A. However, quite many noisy tags make the precision scores average for SpiritTagger (0.35 – 0.62).

8.4.2 Attractions taken from distance

This is images of attractions that are geo-referenced at the exact position of image capture. Image 1B and Image 3 is of this type. These two images are taken and geo-referenced at the exact same position on the Westminster Bridge in London. However, they are captured in opposite directions and are therefore of Big Ben and London Eye respectively.

LoCaTagr performs well, finding 6 relevant tags and only 1 noisy tag for image 1B, and 8 relevant and no noisy tags for Image 3. The unsure tags in Image 1B is based on judgments whether the bridge and river where the image was taken is relevant for the image or not when not being visible on the image. Further, the tag *eye* could and could not be regarded as relevant for Image 3. The precision scores are high both when the unsure tags are regarded as not relevant (0.89 and 0.60) and when they are regarded as relevant (1.00 and 0.90).

LoTagr and SimpleTagr find the exact same set of tags for Image 1B and Image 3 as they are location-based and the location is the same. They get good scores on Image 3 as the river is visible and definitively relevant for the image, whereas it is not given that it is relevant for Image 1B. Although they do get quite high precision scores when unsure tags are regarded as relevant (0.80 and 0.75), it is also important to consider that they only find less than half as many relevant tags as LoCaTagr. Further, they find both Big Ben and London Eye for both images (which ensure one noisy tag for each image).

SpiritTagger also find the same set of tags for Image 1B and Image 3, except that *vacation* is suggested for Image 1B whereas *sky* is suggested for Image 3 (the other 19 suggested tags are similar). This is somewhat surprising, and seems to indicate that SpiritTagger rely more on location than on visual similarity for these images. SpiritTagger performs comparable to LoCaTagr for Image 1B. However, LoCaTagr returns much more relevant tags and fewer noisy tags than SpiritTagger for Image 3. The distance from the bridge to London Eye (Image 3) is bigger than the distance from the bridge to Big Ben (Image 1B). This indicates that it is harder to use location-based information on attractions that are not close to the image capture position without using categories.

8.4.3 Not so famous attraction

Image 4 is an example of these types of attractions that have a limited number of relevant images available because not so many people visit the attraction or take and upload images from it. Image 5 is another example although it will be discussed in the next sub-chapter.

LoCaTagr finds many relevant tags for Image 4, although some tags are describing the same thing only in different languages. This occurs when users have tagged images on Flickr with a tag in two or more different languages (discussed more in Chapter 9). Only one noisy tag is found resulting in high precision scores (0.80 and 0.90).

SimpleTagr does not find any relevant tags, whereas LoTagr performs around the same level as LoCaTagr. This seems to indicate that more advanced methods than using just a simple system is needed to handle not so famous attractions. However, the usage of category does not seem to be that helpful since LoTagr performs comparable to LoCaTagr. This is probably because there are few other attractions nearby the church in Image 5.

SpiritTagger suggest only 2 relevant tags and as much as 12 noisy tags leading to low precision scores (0.12 and 0.29). This could indicate that the category approach works better than content-based image analysis when there are few relevant images available on Flickr. This does make sense in that more images are needed from different angles, views, weather and light conditions etc. to support detection with visual similarity with regard to a specific query image (with its specific angle, view, weather condition, light etc.) whereas the category approach can use all of the available images.

8.4.4 Not so famous attraction near famous attraction

These types of attractions are small attractions that are not so famous, and that have a more famous attraction located nearby. Image 5 is taken at an aquarium in Paris named Cineaqua that is located close to the Eiffel Tower.

LoCaTagr finds 5 relevant tags and no noisy tags resulting in maximum precision score (1.00).

LoTagr and SimpleTagr perform very badly on Image 5, finding no relevant tags and many noisy tags. These systems therefore gets minimum precision score (0.00). The problem is that there is so many images from the bigger attraction that tags from the smaller attraction are not selected. This problem is described in Figure 5.1. Thus, location is not sufficient to find relevant tags for these types of images, whereas combining location with the category approach gives good results.

SpiritTagger is able to recognize the color *blue,* but suggest many tags that are relevant for the nearby Eiffel Tower resulting in many noisy tags. Thus, the precision scores are low (0.11 and 0.28). Although SpiritTagger performs slightly better than the systems using only location, it is not comparable to LoCaTagr. This indicates that the category approach is better than using content-based image analysis for not so famous attractions located nearby famous attractions.

8.4.5 Overview images

This type of images is for example overview images of places or landscapes. Image 6 which is an overview image of Paris is an example.

LoCaTagr finds 3 relevant tags and 1 noisy tag. It could be discussed whether *panorama* is a relevant tag. Image 6 is a panorama image but the image could just as well not have been a panorama image (although most overview images are panorama images). Such tags should probably be handled as contextual information as it should be possible to detect panorama images by looking at the dimensions of the image. It is possible that *panorama* therefore should be added to the tag filtering discussed in Chapter 5.4.2.

LoTagr and SimpleTagr use only location and these systems therefore only find information relevant for the location where the image was captured. As previously discussed, information about the image capture location could and could not be regarded as relevant for the image.

SpiritTagger recognizes many visually relevant tags (*sky, blue, clouds, skyline* and *white*) that LoCaTagr is not able to find. SpiritTagger is therefore the system with the best result for this image even though it suggests a few noisy tags. This indicates that using content-based image analysis on these types of images is beneficial. The three relevant tags (*overview, city* and *view*) found by LoCaTagr are directly linked to the category. Therefore, it would be very interesting to combine content-based image analysis with the category approach for these types of images.

8.4.6 Short-lasting events

Short-lasting events are events lasting up to one day. Image 7 is from a U2 concert lasting some hours. Note that *The Claw* is the name of the stage U2 brought along on their tour, and that *Edge* and *Bono* are two of the group members. The concert was held on Camp Nou which is a football stadium where FC Barcelona (often referred to as Barca) play their home matches. Image 8 is from the national day in USA, also known as the Independence Day or simply July 4 / Fourth of July.

LoCaTagr performs well on these short-lasting events. First of all no noisy tags are found. Secondly, 10 and 3 relevant tags are found for Image 7 and 8 respectively. The precision scores are therefore perfect (except if *nou* and *camp* are regarded as noisy for the image from *camp nou*).

LoTagr and SimpleTagr finds a few relevant tags (1 to 3) but always finds at least as many noisy tags resulting in low precision scores. Further, the relevant tags found by these location-based systems are directly related to the location only and not the event itself. Therefore, many tags related to football are found for Image 7 because football is the main activity performed at the stadium.

SpiritTagger do not suggest any tags for Image 7 for unknown reasons. It suggests 3 relevant tags for Image 8. It is the only system suggesting *Independence Day*. It also recognizes that the image is taken at *night*. Further, *monument* is suggested. However, it seems to relate the image to war and more specifically World War 2. These and other noisy tags make the precision scores low (0.18 and 0.35). It is possible that combining the category approach with content-based image analysis could make it possible to discard the war tags as noisy.

8.4.7 Long-lasting events

Long-lasting events are events lasting for several days and up to around a month. Image 9 is from the yearly Roskilde Festival in Denmark. Image 10 is from the 2010 Winter Olympics in Vancouver.

LoCaTagr finds 8 relevant tags for both these images. No noisy tag is found for Image 10 whereas 1 noisy tag is found for Image 9. The precision scores are therefore high (1.00 and 0.89).

LoTagr performs quite well on these images. However, it finds the tag *roskildefestival2006* for Image 9 (which is from the Roskilde Festival 2009). Further, it finds tags not related to the Olympics in Image 10 (*football*, *cfl* and *bclions*). BC Lions is a football team playing in the Canadian Football Leage (CFL). They play their home matches at BC Place Stadium, which hosted the closing ceremony of the Olympic Games. This indicates once again that the category approach limits the number of noisy tags.

SimpleTagr finds 33 and 5 tags for Image 9 and Image 10 respectively. For Image 9, many relevant tags are found but more than double as many noisy tags are also found. For Image 10 only one noisy tag is found but no relevant tags are found. The big differences in number of tags found indicate that more advanced methods are preferable. This is backed up by the results from LoCaTagr and LoTagr which is much more stable (finds around the same amount of tags for both images).

SpiritTagger only find one relevant tag (*summer*) for the festival image and no relevant tags for the image from the Olympics. It should however be noted that it probably is difficult to find any relevant tags by looking at the content of these images as they are rather dark and with no obvious parts of the image standing out visually.

8.4.8 More on events

The information found by the image tagging systems for some of the events is limited. For example, Lily Allen is the artist performing on the image from the Roskilde Festival. Further, the closing ceremony is not listed as a tag for the image from the closing ceremony of the Olympics. This is not surprising because these images are categorized as long-lasting events. However, categorizing them as short-lasting events do not lead to finding tags related to Lily Allen or the closing ceremony. The problem is that there are not enough images available from these short-lasting events that are part of the bigger long-lasting events.

It should be noted that finding or suggesting tags for events are generally much harder than for objects. The problem is basically that there are limited numbers of images available from events. The process of tagging events should not be harder than tagging objects as long as there are enough images available for the events. In fact, it should be easier to restrict the occurrence of noisy tags as the time intervals used are short. Therefore, images inside the time interval are very likely to be relevant for the query image.

Events must be quite big before there are enough geo-referenced and tagged images on Flickr available from the event to get decent results. LoCaTagr require at least around 10 usable images from different users to produce a good set of relevant tags. The given examples discussed in 8.4.6 and 8.4.7 illustrates that LoCaTagr is able to tag images from events successfully as long as enough images are available. However, many other events will not give as good results because there are not enough images available.

It seems that LoCaTagr is able to deal with images from events better than SpiritTagger. One of the reasons could be that images from events vary a lot in content, i.e. the images could visually be very different but still relevant as long as they are from the same event. Therefore, content-based image analysis requires more similar images to be able produce relevant tags than the category approach requires since it can use all images regardless of visual similarity.

8.4.9 Importance of correct sub-category

It is interesting to investigate the importance of the name of the sub-categories for areas with many images available. Using different sub-categories will lead to different results as different sets of images are found in the result set. This is because the sub-category is used as one of the search parameters. This can be seen in Table 8.1. The table displays the tags found by using different sub-categories for Image 1B. All of the sub-categories belong to the main category object. Place names are not shown in the table.

Note that usage of categories such as *double-decker* and *bus* (or *cab* and *taxi*) will lead to the same results because they are synonyms and LoCaTagr use all synonyms of the sub-category in the image search process as described in Chapter 5.2.7.

The three categories *tower, clock* and *clock tower* all find the same set of 6 very relevant tags (*bigben*, *tower, clock*, *clocktower*, *parliament*, *housesofparliament*). By using the category *clock tower*, the tag *architecture* is found

Category / **Tag**	Tower	Clock	Clock Tower	Big Ben	Parlia-lia-ment	Palace	Bus	Cab	Bridge	Hotel
bigben	X	X	X	X	X	X	X	X	X	
tower	X	X	X							
clock	X	X	X							
clocktower	X	X	X							
big				X			X			
ben				X			X			
parliament	X	X	X	X	X	X		X		
housesofparliament	X	X	X		X	X		X		
palaceofwestminster						X				
westminsterpalace						X				
bridge							X		X	
westminsterbridge	X					X			X	
river									X	X
thames	X		X			X			X	X
riverthames	X					X				
taxi								X		
bus							X			
red							X			
architecture			X							
night	X						X			
londoneye									X	X
countyhall										X
hotel										X

Table 8.1 – Tags found by LoCaTagr for Image 1B with different sub-categories. All the sub-categories belong to the main category *object*. Place names are not shown in the table.

in addition. By using the category *tower*, the noisy tag *night* is found along with tags relevant for the position of image capture (the bridge and river). It can be discussed which category that is best to use because all the three categories returns good results. Using more general categories such as *tower* is preferable as it allows re-usage of categories (e.g. same category can be used for Eiffel Tower).

The category *Big Ben* is not desirable to use because the actual names of the attractions, places and events are too detailed, and therefore the categories will seldom be re-used. Further, the results with *Big Ben* as category is not good compared to the three categories discussed in the last paragraph.

When using the sub-category *parliament*, only three tags are found. However, the tags found are very relevant for the sub-category and therefore this might be a good choice of sub-category if the image were more centered towards the middle of the parliament building (i.e. if the Houses of Parliament was the main subject of the image rather than Big Ben).

The Houses of Parliament is also referred to as both Palace of Westminster and Westminster Palace. The category *palace* can therefore also be a natural choice of image category (but again, more suitable if the image was centered more towards the palace / parliament building). The other two names of the parliament building are found by using this category. Further, information about the bridge and river is found. It can therefore be discussed which of the two categories *parliament* or *palace* that would be the best option if the image was more centered towards the castle / parliament building.

As can be seen from Table 8.1, *bus* and *taxi* are only found if the sub-category is *bus* or *cab* (*taxi*) respectively. *Red* is returned when using the sub-category *bus*, which is a relevant tag since the bus on the image is red (as most buses in London). These findings indicate that using several categories might be beneficial. However, this is probably only reasonable if automatic categorization is implemented as it perhaps would require too much from users to select multiple categories. Another possibility is to make it optional to specify more categories. This has not been explored in this work.

When using the sub-category *bridge*, almost only tags relevant for the bridge and river are returned. Thus, the sub-category *bridge* would be a good choice if the main subject of the image was the Westminster Bridge. The tags *bigben* and *londoneye* had low frequencies and were almost filtered out. It is likely that either Big Ben or London Eye is visible on images of the bridge (as they are located on either side of the bridge). Thus, one of these two tags is probably relevant for most images of the bridge (while the other is noisy).

Finally, it is worth mentioning that using wrong categories will result in that no relevant tags are found. The last sub-category in the table, *hotel*, is an example. Further, typing errors in the sub-categories (e.g. *towre* or *cluck*) will lead to not finding any tags.

8.4.10 Images with same location

Image 1A and 1B is the same image but geo-referenced differently. Image 1A is geo-referenced at the exact position of Big Ben whereas Image 1B is geo-referenced at the exact position of image capture (Westminster Bridge).

Image 1B and Image 3 are geo-referenced at the exact same position on the Westminster Bridge but are taken in opposite directions and are thus images where Big Ben and London Eye is the main subject respectively.

The results from LoCaTagr for this set of images (1A, 1B and 3) are summarized in the Table 8.2. Place names are not included.

As can be seen from Table 8.2, LoCaTagr is able to find many relevant tags for all the three images with only one noisy tag for one of the images. The systems using only location (LoTagr and SimpleTagr) returned few relevant tags (1-3) for these images.

These results are very encouraging and indicate that using categories is very useful for automatic image tagging systems, both to support geo-referencing at the exact position of image capture and to differentiate between attractions located at the same place.

Image 1A	Image 1B	Image 3
bigben	bigben	londoneye
tower	tower	ferriswheel
clock	clock	wheel
clocktower	clocktower	millenniumwheel
parliament	parliament	river
big	housesofparliament	thames
ben	westminsterbridge	riverthames
	thames	eye
	riverthames	
	night	

Table 8.2 – Comparison of tags found by LoCaTagr for Image 1A, 1B and 3. Place names are not shown in the table. Image 1A and 1B is the same image but geo-referenced differently. Image 1B and 3 is geo-referenced at the same location but is not the same image.

8.4.11 Runtime comparison

The runtime of LoCaTagr and LoTagr will vary depending on workload on server, workload on Flickr, workload on Synonyms.net (only for LoCaTagr), number of useable images found and finally number of non-usable images found. The runtimes (and list of tags) can therefore also vary when processing the same image twice.

Because of the dynamic approach for deciding how many images to use (Chapter 5.1), the runtimes will vary depending on how many useable and non-usable images that are in the result set. Non-usable images are images from users that already have images in the result set (Chapter 5.3) and images with one or zero tags (Chapter 5.4.1). Many non-usable images will make the runtimes longer. The runtimes will also be longer for query images that have few relevant images available because more search requests are needed to find relevant images (by increasing radius as shown in Table 5.2). This can be seen in Figure 8.3 for LoCaTagr.

Figure 8.3 shows that the runtimes for LoCaTagr are shorter than average for Image 1, 2, 3 and 6 that are from areas with many images compared to the other images that have fewer relevant images available nearby. The runtime of Image 9 and 10 for LoCaTagr is much longer than average. The reason is that many images from the same set of a few users are available on Flickr for these images, but overall few images from unique users are available. For example, for Image 10, a total of 789 images are processed, but only 24 of these are used by LoCaTagr.

SimpleTagr is by far the fastest system. The runtimes of this system is also very stable (4 to 7 seconds). This is because the system only performs one search request to Flickr, and tag processing is very limited.

The runtimes of SpiritTagger are only approximates since they are measured manually using its web interface. However, the results indicate that Spirit-Tagger is a few seconds faster than LoCaTagr on average. This can be seen in Figure 8.4 with the average run-time of LoCaTagr to the left and the average run-time of SpiritTagger to the right.

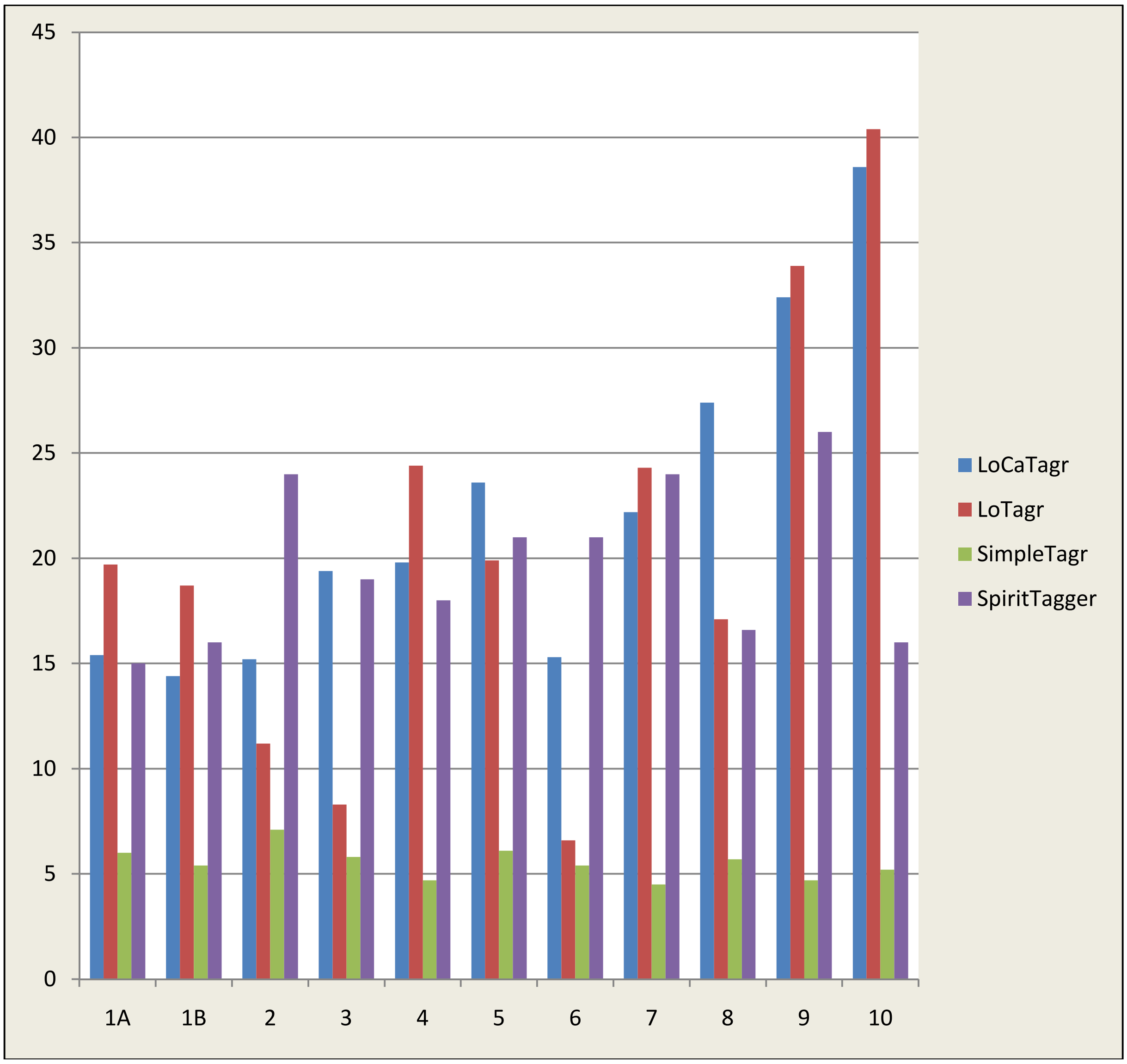


Figure 8.3 – Runtime in seconds for LoCaTagr and the three comparison systems for the images discussed in the evaluation (Image 1A – Image 10). The runtimes can vary with several seconds for each time the systems are run. Further, the measurements of SpiritTagger are only approximates.

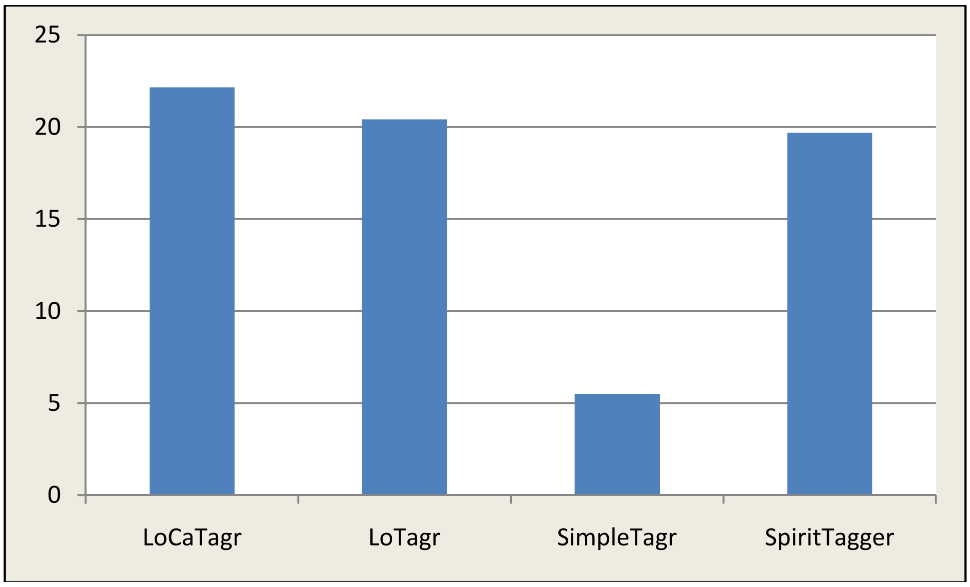


Figure 8.4 – Average runtime in seconds for LoCaTagr and the three comparison systems for Image 1A - 10. The measurements of SpiritTagger are only approximates.

8.4.12 Place names

As discussed in Chapter 4.4, place names can vary in accuracy from continent to neighborhood or street name. Place names are therefore not discussed and evaluated in detail in this work. The results indicate that all the four systems are able to tag the images with mostly correct place names. The systems also finds around the same amount of place names averagely. This can be seen in Figure 8.5 which is an overview of the different types of tags found by LoCaTagr and the three comparison systems (the blue parts of the bars representing place names are roughly equally big). Figure 8.6 displays the results of Figure 8.5 in percentage with regards to the total number of tags found by each system.

8.4.13 Summary

The number of tags found by each of the four systems is summarized in Table 8.7, and the average is listed in Table 8.8. The results are also shown graphically in Figure 8.5, 8.6, 8.9, 8.10 and 8.11.

LoCaTagr finds around 6 relevant tags for each image on average. Further, a noisy tag is found averagely in every 0.36 image (or around 1 noisy tag in each image if the unsure tags are regarded as noisy). From Table 8.7 it is also noticeable that at least 3 relevant tags are found for each image and that no more than one noisy tag is found for the same image. Five of the eleven images have neither noisy nor unsure tags resulting in perfect precision scores.

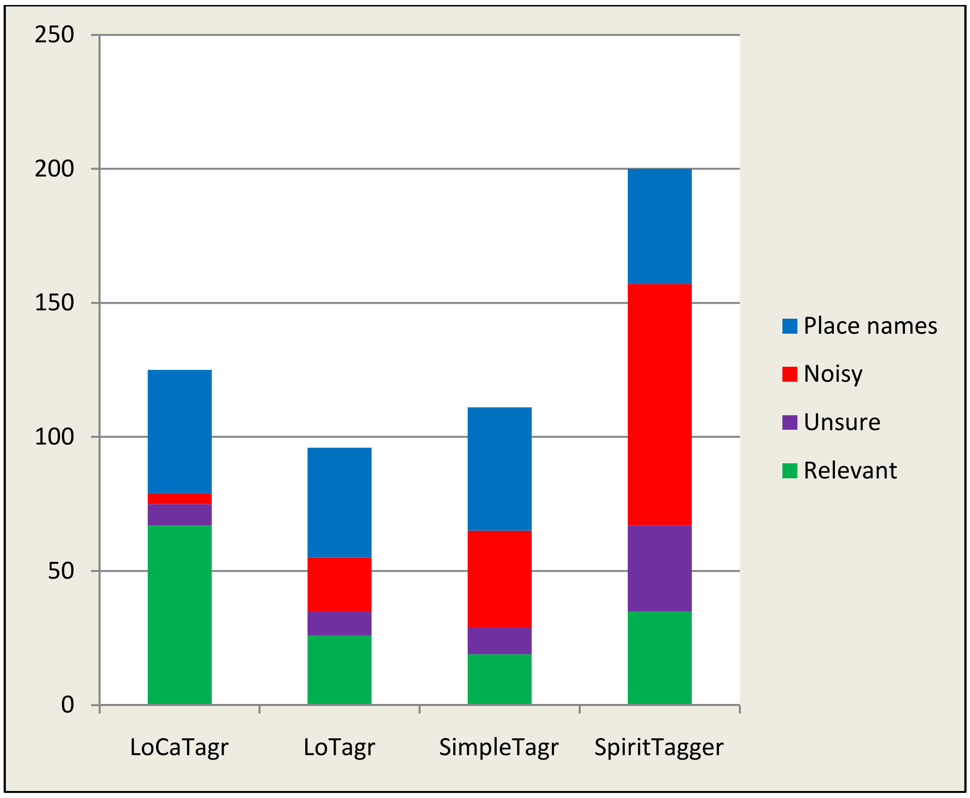


Figure 8.5 – Overview of different types of tags found by LoCaTagr and the three comparison systems for Image 1 – 10.

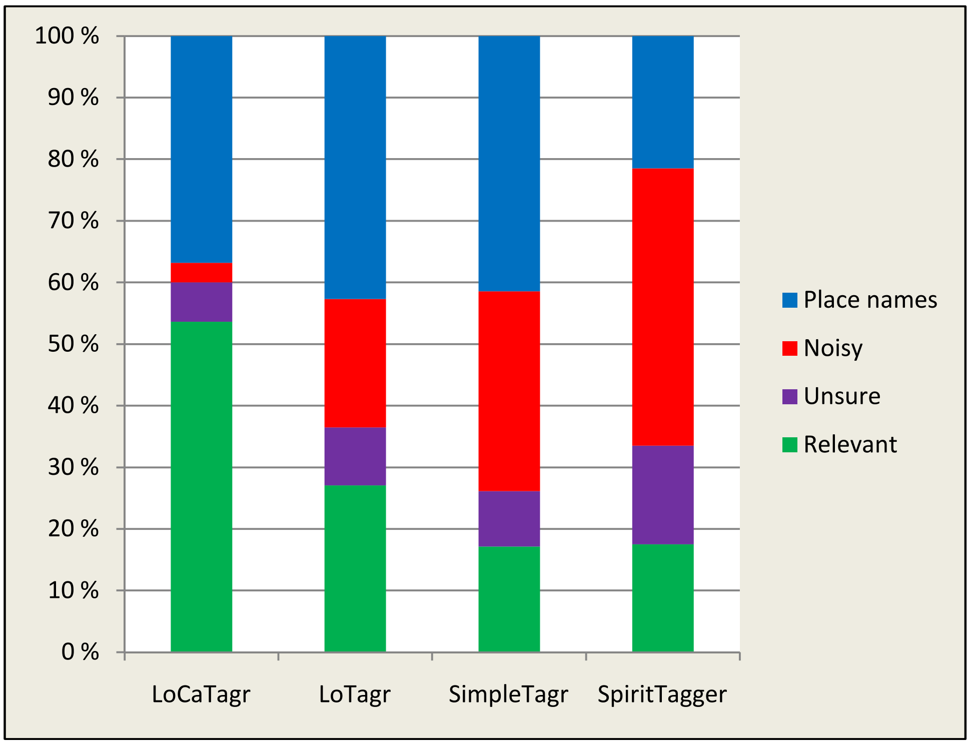


Figure 8.6 – Percentage of different types of tags found by LoCaTagr and the three comparison systems for Image 1 – 10.

LoTagr generally finds fewer tags per image than LoCaTagr (8.73 compared to 11.36). However, it finds 1.82 noisy tags per image compared to 0.36 in LoCaTagr. Further, only 2.36 relevant tags per image are found by LoTagr compared to 6.09 in LoCaTagr. This remarkably better performance by LoCaTagr compared to a similar system not using categories is a strong indication that usage of categories is beneficial for automatic image tagging systems.

SimpleTagr performs worse than LoTagr. This is not surprising as LoTagr use more advanced methods (e.g. tag filtering, restrict usage from same users). Another thing to take notice of concerning SimpleTagr is that the number of tags found varies a lot between each image (from 4 to 33) as can be seen in Table 8.7 and Figure 8.9.

LoTagr and SimpleTagr have precision scores at 0.00 for some images, which mean that they do not find any relevant tags for these images. This can be seen in Figure 8.10 and 8.11, which are graphical overviews of the two precision scores.

It must be mentioned that some of the tags found by LoCaTagr (and LoTagr and SimpleTagr) are representing the same thing only in different languages. However, this only applies to Image 4, 5 and 7 which are taken in countries where English is not the native language. Further, this also applies similarly to SpiritTagger in Image 5.

SpiritTagger gets the lowest precision scores of all systems as shown in Figure 8.10 and 8.11. However, as previously discussed, the calculation of precision scores are not fair to SpiritTagger as it always suggests exactly 20 tags for each image and is a system suggesting tags rather than assigning tags automatically.

Therefore, it can be more interesting to look at number of relevant tags found or suggested averagely per image (Table 8.8). SpiritTagger suggests more than double as many relevant tags as SimpleTagr for each image averagely, and around 48 percent more than LoTagr. However, even though SpiritTagger on average nearly suggest double as many tags as LoCaTagr finds, it is only able to suggest 3.50 relevant tags averagely per image compared to the 6.09 relevant tags found with LoCaTagr. This is also shown graphically in Figure 8.5 and 8.6.

However, SpiritTagger were able to suggest some interesting tags for some of the images that the other systems did not find (e.g. *architecture*, *sky* and *City*). These tags are probably found from visually similar images and shows that the use of content-based image analysis is useful. It would therefore be very interesting to combine the two approaches. It is possible that the category approach could lower the high noise rate in SpiritTagger, and assist in selecting relevant tags.

Finally it should be mentioned that only a small set of images is tested and evaluated in this thesis. The images used in the evaluation are only examples. However, results from other images have shown to give similar results.

Type	Image System	**1A**	**1B**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Total	**LoCaTagr**	12	15	5	16	14	11	8	14	6	11	13
	LoTagr	7	10	5	10	9	7	5	10	9	12	12
	SimpleTagr	6	8	9	8	4	11	4	17	6	33	5
	SpiritTagger	20	20	20	20	20	20	20	-	20	20	20
Place names	**LoCaTagr**	5	5	2	8	4	6	3	4	3	2	4
	LoTagr	4	5	2	5	5	4	4	2	4	2	4
	SimpleTagr	4	4	3	4	3	4	3	9	3	5	4
	SpiritTagger	7	7	3	7	3	2	2	-	3	5	4
Rele-vant	**LoCaTagr**	5	6	3	7	8	5	3	10	3	8	9
	LoTagr	1	1	3	3	4	0	0	1	2	7	4
	SimpleTagr	1	1	3	3	0	0	0	3	1	7	0
	SpiritTagger	5	6	6	4	2	2	6	-	3	1	0
Unsure	**LoCaTagr**	2	3	0	1	1	0	1	0	0	0	0
	LoTagr	2	3	0	1	0	0	1	2	0	0	0
	SimpleTagr	1	2	0	0	0	0	1	0	0	6	0
	SpiritTagger	3	5	3	1	3	3	5	-	3	3	3
Noisy	**LoCaTagr**	0	1	0	0	1	0	1	0	0	1	0
	LoTagr	0	1	0	1	0	3	0	5	3	3	4
	SimpleTagr	0	1	3	1	1	7	0	5	2	15	1
	SpiritTagger	5	2	8	8	12	13	7	-	11	11	13
Preci-sion1	**LoCaTagr**	0.71	0.60	1.00	0.88	0.80	1.00	0.60	1.00	1.00	0.89	1.00
	LoTagr	0.33	0.20	1.00	0.60	1.00	0.00	0.00	0.13	0.40	0.70	0.50
	SimpleTagr	0.50	0.25	0.50	0.75	0.00	0.00	0.00	0.38	0.33	0.25	0.00
	SpiritTagger	0.38	0.46	0.35	0.31	0.12	0.11	0.33	-	0.18	0.07	0.00
Preci-sion2	**LoCaTagr**	1.00	0.90	1.00	1.00	0.90	1.00	0.80	1.00	1.00	0.89	1.00
	LoTagr	1.00	0.80	1.00	0.80	1.00	0.00	1.00	0.38	0.40	0.70	0.50
	SimpleTagr	1.00	0.75	0.50	0.75	0.00	0.00	1.00	0.38	0.33	0.46	0.00
	SpiritTagger	0.62	0.85	0.53	0.38	0.29	0.28	0.61	-	0.35	0.27	0.19

Table 8.7 – The number of different types of tags and precision scores for each system and image. These results are shown graphically in Figure 8.9, 8.10 and 8.11.

Type System	**Total tags**	**Place names**	**Rele-vant**	**Unsure**	**Noisy**	**Preci-sion1**	**Preci-sion2**
LoCaTagr	11.36	4.18	6.09	0.73	0.36	0.86	0.95
LoTagr	8.73	3.73	2.36	0.82	1.82	0.44	0.69
SimpleTagr	10.09	4.18	1.73	0.91	3.27	0.27	0.47
SpiritTagger	20	4.30	3.50	3.20	9.00	0.23	0.44

Table 8.8 – The average (arithmetic mean) of the different types of tags for each system. SpiritTagger did not suggest any tags for Image 7. Therefore the average is based on 10 images for SpiritTagger and on 11 images for the other systems.

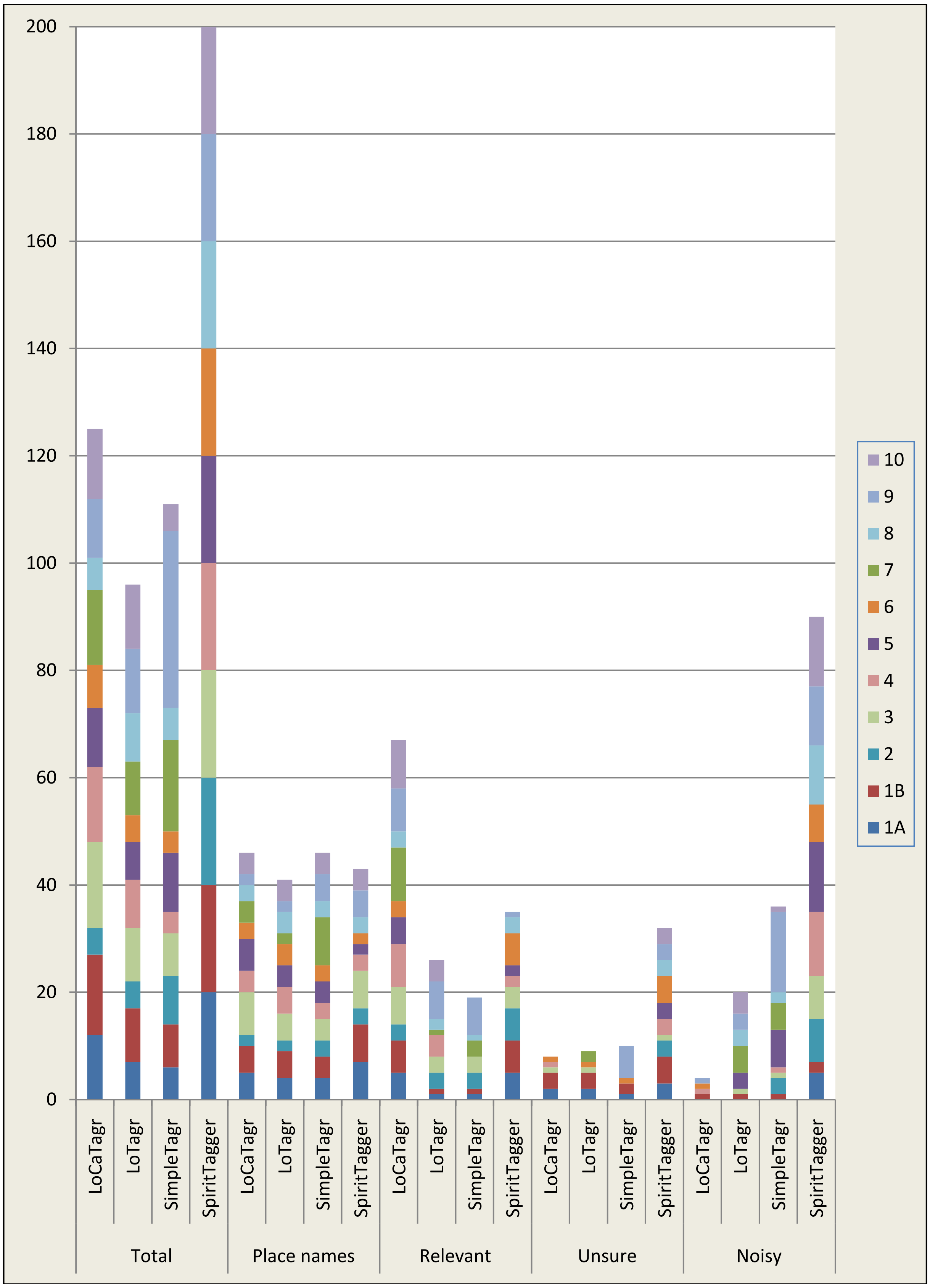


Figure 8.9 – Graphical overview of number of tags for LoCaTagr and the three comparison systems for Image 1A - 10. The numbers to the left are number of tags.

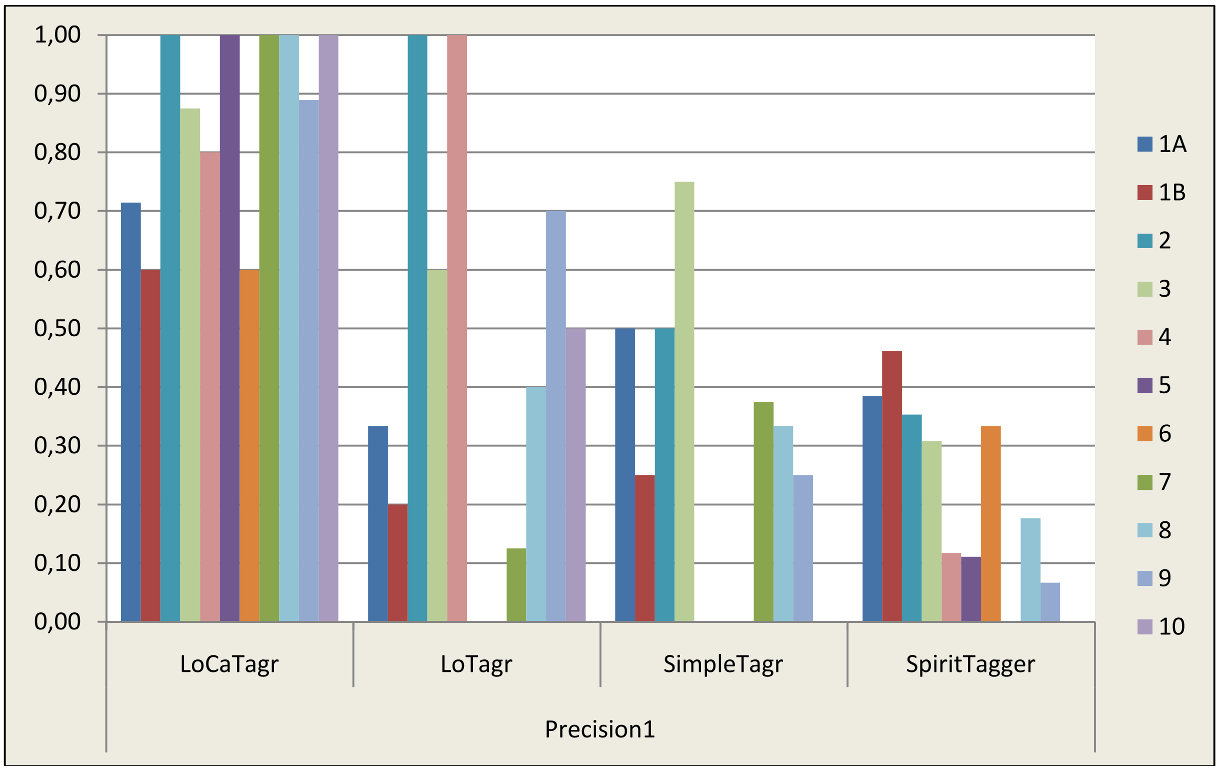


Figure 8.10 – Graphical overview of Precision1 for LoCaTagr and the three comparison systems. The unsure tags are regarded as noisy.

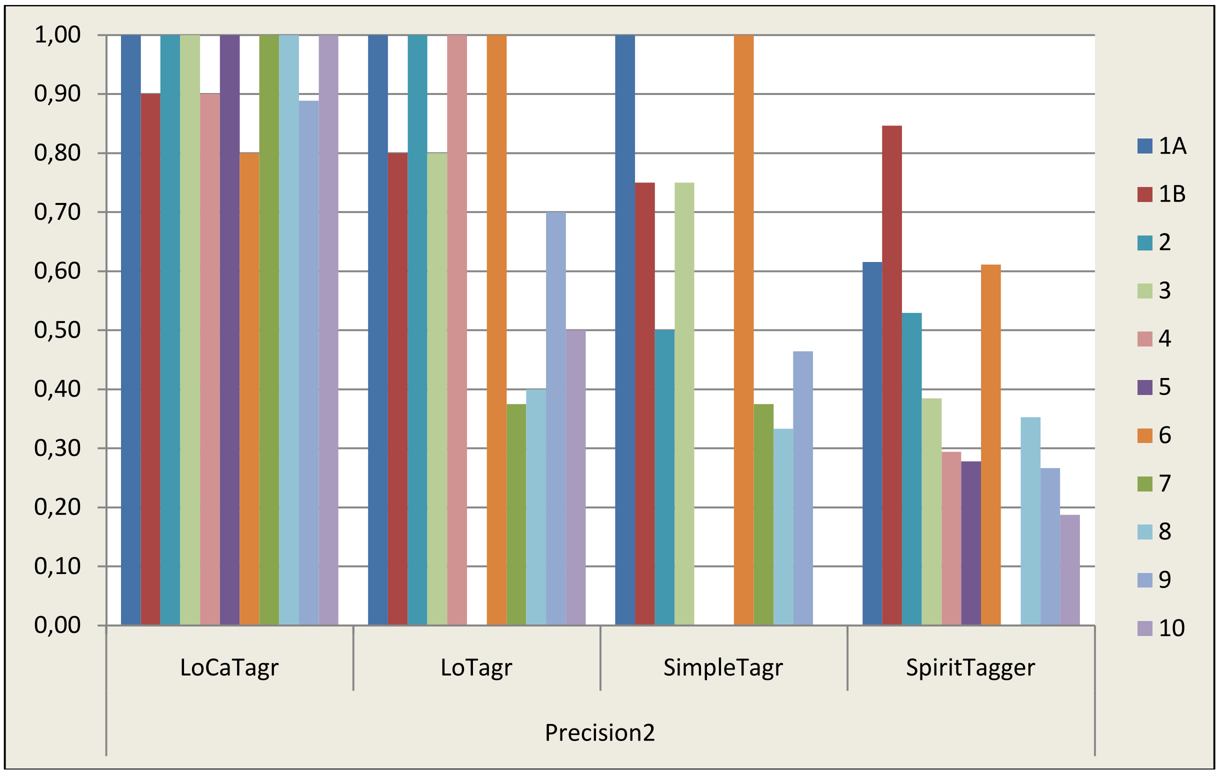


Figure 8.11 – Graphical overview of Precision2 for LoCaTagr and the three comparison systems. The unsure tags are regarded as relevant.

Chapter 9

Future Work

In this chapter, some possible future work will be discussed.

Content-based image analysis could as already discussed be interesting for detecting tags that can be seen visually on the image. SpiritTagger is able to suggest some interesting tags (*night* in Image 8, *blue* in Image 5 and some tags such as *architecture*, *sky* and *city* in Image 1B, 2, 3 and 6) that most likely are found by using visual similarity techniques. LoCaTagr is only able to tag things that are already tagged on relevant images nearby, i.e. it is not able to tag special things occurring on images such as the taxi in Image 1A and 1B. It would therefore be interesting to extend the category approach using location and date/time to also take use of content-based image analysis. It could be possible that content-based image analysis is more usable for some categories than other categories, and therefore the system can put more weight on content-based image analysis for these kinds of images. The evaluation indicated that the system using content-based image analysis performed better than LoCaTagr for overview images (Image 6).

Another interesting approach to investigate would be to use several categories for the same image. This would be especially interesting for Image 1 (both 1A and 1B) as use of all the categories *tower*, *bus* and *taxi* for this image could lead to many relevant tags as shown in Table 8.1. However, asking the users to provide several categories might not be the best option as it would require much from users of the system (although it could be optional). Therefore, automatic categorization (discussed in next paragraph) seems like a requirement for using several categories.

Automatic categorization would prevent the need for users to specify categories. It could be based on content-based image analysis and/or on the work in [16] which is able to categorize *tags*. A possibility would then be to categorize an *image* based on the categories of its most frequent *tags*. Place names could have to be discarded by using external sources such as GeoNames as they are probably not convenient to categorize. Further, georeferenced articles or other external sources could possibly be used to find out that for example Big Ben is a tower (as Big Ben is likely to be the most frequent tag for images nearby the tower). It is interesting to see that LoTagr (which do not use categories) could have been able to categorize some of the images discussed in the evaluation by taking the most frequent tag that is not a place name (Image 1A, 2, 3, 4 and 9). However, problems arise when the image is geo-referenced at distance from the attraction (Image 1B and 6), when there are many attractions at the same place (Image 5), and for events happening at places where other type of events also occurs (Image 7, 8 and 10). Therefore, more advanced methods would be needed and overall this might be very difficult to solve.

LoCaTagr does not take use of external information sources regarding weather information and place names. It is possible that categories can be helpful when using this kind of information. For example, weather information can be more relevant for certain categories than for other categories. Further, the accuracy of location names could possibly be handled differently based on categories (e.g. natural for one category to use street name whereas another category should rather use country name instead or in addition to street name).

Another possible extension of LoCaTagr is to use tag ranking instead of using just the most frequently used tags. Taking the most frequent tags could just give the most commonly used tags, which are not necessarily the best or most relevant tags for an image. Therefore, use of some sort of ranking method or algorithm to favor relevant tags and/or prevent unrelated tags from being used could be interesting. Systems using visual similarity techniques usually give some kind of score of how similar the images are to the query image. Images with low similarity score are likely to not be relevant and the tags from these images can be discarded. In this work, the three metadata sources available to work with are category, date/time and location. Category is impossible or hard to rank because it makes little sense to rank based on category as it is difficult to decide whether one category is better or more relevant than another. However, a ranking system favoring the images that are closest to the query image both in location and time could be interesting.

Information about altitude and compass/direction could be used to favor images taken around the same altitude and in the same direction as the query image higher than images with different altitude and direction. This could for example make it possible to distinguish images taken at the top of Eiffel Tower or a cliff from images taken at the bottom of Eiffel Tower or a cliff. Further, images taken at the exact same position in the same direction can be regarded as more relevant than images taken in the opposite direction. This would be very useful for Image 1B and Image 3. It would require a magnetic compass, and that the cardinal point and altitude gets stored in the EXIF-header together with the GPS coordinates (latitude and longitude).

Language can be a problem for images taken in countries where English is not the native language. This can occur if users have tagged images on Flickr with a tag in two (or more) different languages as the results for Image 4 and 5 demonstrate. The question is then if both tags should be used, or which language to use. Some way to translate tags would be needed to solve this problem, and some way to remove duplicate tags (e.g. *blue* and *bleu*). But it should also be possible for French users to search for *bleu*. This is a complicated problem.

Other implementation specific things that could be improved:

- Better user interface for categorizing images. For example it should be possible to categorize several images in one operation.
- Saving the tags found with LoCaTagr to the EXIF-header of the image instead of just displaying them in a table on the web interface. Displaying the tags in a table on the web interface is chosen for now as the system is only a prototype for testing purposes.
- Allow the possibility to manually geo-reference images on a map like in Flickr and Panoramio.
- Use more sources such as Panoramio in addition to Flickr (for finding relevant images and tags).
- Allow the possibility of manually specifying the event categories more detailed. For example specify exactly how long the event is and when it is, and whether the event is repeated or not. This would also require that LoCaTagr takes use of the provided information.
- Optimize the implementation with regards to performance (runtime). The runtime can probably be lowered significantly as the performance have not been prioritized.

Chapter 10

Conclusion

In this master thesis, LoCaTagr, a location, category and time-based automatic image tagging system using Flickr, have been designed and implemented. LoCaTagr is able to find relevant tags for a query image as long as there are a sufficient number of geo-referenced and already tagged images available on Flickr that is relevant for the query image. The query image must be geo-referenced, have date/time of image capture available, and the user must provide an image category.

LoCaTagr handles images based on which category the images belongs to, i.e. it is handled the best way to handle images in that specific category. This means that images of objects or places are handled differently than images from events.

The evaluation demonstrates that LoCaTagr finds very few noisy tags (0.36 per image) despite using a noisy image database. The noise level is kept low because the category approach restricts the usage of images that are not relevant. Further, usage of tags from same users is restricted and the system utilizes a dynamic approach for using many images when possible, and fewer images when not many relevant images are found.

The evaluation further indicates that LoCaTagr is able to find around 3 to 10 relevant tags for each image. LoCaTagr generally performs very good compared to baseline image tagging systems using only location. It performs especially good compared to systems using only location for events, for images geo-referenced at distance from the attraction, and when there are other attractions located nearby the attraction on the image.

It also performs very good compared to SpiritTagger, which use both location and content-based image analysis. However, content-based image analysis proved to be useful for detecting certain tags such as *architecture*, *sky* and *city*. This shows that visual similarity techniques can be used to detect tags which are not necessarily linked directly to the image category or location. It would therefore be very interesting to combine the category approach with content-based image analysis to make even better automatic image tagging systems.

The hypothesis was that "*using categories together with location and date/time will result in more relevant and less non-relevant tags than by using other approaches*". This work has shown that using image categories is beneficial when tagging images compared to baseline systems using only location. The category approach also works good compared to a system using content-based image analysis. This thesis is therefore an indication that the hypothesis can be verified, although there might exist other approaches that work at a similar or better level.

References

1. Datta, R., Dhiraj, J., Li, J., Wang, J. Z., *Image retrieval: Ideas, influences, and trends of the new age.* ACM Computing Surveys, 2008. Vol. 40(No. 2): p. Article 5.

2. Karlsen, R. and B. Jakobsen, *Image centric information collection.* To be published, 2010.

3. Jakobsen, B., *Collecting relevant images context information.* Master's Thesis in Computer Science at Faculty of Science and Technology, Department of Computer Science, University of Tromsø, 2010.

4. Ames, M. and M. Naaman, *Why we tag: Motivations for annotation in mobile and online media.* In proceedings of the SIGCHI conference on Human Factors in computing systems (CHI 2007), 2007.

5. Dey, A.K., *Understanding and Using Context.* Personal Ubiquitous Comput. 5, 1, 2001: p. 4-7.

6. Arnold, W.M.S., *Content-Based Image Retrieval at the End of the Early Years.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000. 22: p. 1349-1380.

7. Moxley, E., J. Kleban, and B.S. Manjunath, *Spirittagger: a geo-aware tag suggestion tool mined from flickr*, in *Proceeding of the 1st ACM international conference on Multimedia information retrieval*. 2008, ACM: Vancouver, British Columbia, Canada. p. 24-30.

8. Wang, X.-J., et al., *AnnoSearch: Image Auto-Annotation by Search*, in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*. 2006, IEEE Computer Society. p. 1483-1490.

9. Naaman, M., et al., *Context data in geo-referenced digital photo collections*, in *Proceedings of the 12th annual ACM international conference on Multimedia*. 2004, ACM: New York, NY, USA. p. 196-203.

10. Ahern, S., et al., *ZoneTag: Designing Context-Aware Mobile Media Capture to Increase Participation.* Proceedings of the Pervasive Image Capture and Sharing: New Social Practises and Implications for Technology Workshop at the Eight International Conference on Ubiquitous Computing (UbiComp 2006), 2006.

11. Larsen, J.E. and M. Luniewski, *Using mobile phone contextual information to facilitate managing image collections.* {PIM} 2009 : Personal information intersections: What happens when {PIM} spaces overlap?, 2009: p. 73 - 75.

12. Smeulders, A.W., Worring, M., Santini, S., Gupta, A., and Jain, R., *Content-based image Retrieval at the End of the Early Years.* IEEE Trans. Pattern Anal. Mach. Intell. 22, 12, 2000: p. 1349-1380.

13. Rosch, E., et al., *Basic objects in natural categories.* Cognitive Psychology, 1976. 8(3): p. 382-439.

14. Miller, G.A., *WordNet: A Lexical Database for English.* Communications of the ACM, 1995. Vol. 38(No. 11): p. 39-41.

15. Popescu, A., et al., *MonuAnno: automatic annotation of georeferenced landmarks images*, in *Proceeding of the ACM International Conference on Image and Video Retrieval*. 2009, ACM: Santorini, Fira, Greece. p. 1-8.

16. Rattenbury, T., N. Good, and M. Naaman, *Towards automatic extraction of event and place semantics from flickr tags*, in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. 2007, ACM: Amsterdam, The Netherlands. p. 103-110.

17. Wang, C., D. Blei, and L. Fei-Fei, *Simultaneous image classification and annotation.* IEEE Conference on Computer Vision and Pattern Recognition 2009: p. 1903-1910.

18. Sigurbjornsson, B. and R.v. Zwol, *Flickr tag recommendation based on collective knowledge*, in *Proceeding of the 17th international conference on World Wide Web*. 2008, ACM: Beijing, China. p. 327-336.

19. Graham, A., et al., *Time as essence for photo browsing through personal digital libraries*, in *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. 2002, ACM: Portland, Oregon, USA. p. 326-335.

20. Quack, T., B. Leibe, and L.V. Gool, *World-scale mining of objects and events from community photo collections*, in *Proceedings of the 2008 international conference on Content-based image and video retrieval*. 2008, ACM: Niagara Falls, Canada. p. 47-56.

21. Kennedy, L.S., S.-F. Chang, and I.V. Kozintsev, *To search or to label?: predicting the performance of search-based automatic image classifiers*, in *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*. 2006, ACM: Santa Barbara, California, USA. p. 249-258.

22. Pros, B., J. Schoning, and A. Kruger, *iPiccer: automatically retrieving and inferring tagged location information from web repositories*, in *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*. 2009, ACM: Bonn, Germany. p. 1-2.

Appendix A – List of images

Images used to test and evaluate LoCaTagr:

- Image 1: http://www.flickr.com/photos/mariosp/3641861357/
- Image 2: http://www.flickr.com/photos/97964364@N00/335307270/
- Image 3: http://www.flickr.com/photos/maong/1962604968/
- Image 4: http://www.flickr.com/photos/aikijuanma/163201333/
- Image 5: http://www.flickr.com/photos/baro/4288426933/
- Image 6: http://www.flickr.com/photos/ijansch/3094101740/
- Image 7: http://www.flickr.com/photos/adriagarcia/3697657996/
- Image 8: http://www.flickr.com/photos/metalchris/3689265502/
- Image 9: http://www.flickr.com/photos/auravox/3701612535/
- Image 10: http://www.flickr.com/photos/thelastminute/4398615662/

All the images used are under a Creative Commons license[29] allowing usage, editing and distribution as long as the source is listed.

Note that GPS coordinates and date/time have been added to the EXIF-header of these images. Therefore, the manually inserted values might not correlate with the actual location and time of image capture.

The images listed above (with added GPS coordinates and date/time) can be used as example images on the web interface. The images are named as following:

- Image 1A: big_ben.jpg
- Image 1B: big_ben2.jpg
- Image 2: eiffel_tower.jpg
- Image 3: london_eye.jpg
- Image 4: ishavskatedralen.jpg
- Image 5: cineaqua.jpg
- Image 6: paris_overview.jpg
- Image 7: u2_camp_nou.jpg
- Image 8: independence_day.jpg
- Image 9: roskilde_2009.jpg
- Image 10: olympics_2010.jpg

Note that the image names are only for convenience. LoCaTagr does not take advantage of the information provided in the image names.

[29] http://creativecommons.org/licenses

Appendix B – LoCaTagr code

The LoCaTagr code follows. API keys and secrets are removed and replaced with *(removed)* in the code.

```
import EXIF
import datetime
import sys
import unicodedata
import flickrapi
import urllib
import time

INITIAL_REQ_IMAGES = 50     # The initial number of required images in result set
LIMIT_FACTOR = 10           # The limit factor of required images
START_RADIUS = 0.001        # The radius to start the image search with
RADIUS_MULTIPLIER = 2.0     # The multiplier used to increase radius
MAX_FLICKR_RADIUS = 32      # The maximum allowed radius in Flickr
FREQUENCY_LIMIT = 0.20      # The limit on how big the frequency must
                            # be for a tag to be considered relevant

TAG_FILTER = ("geotagged", "", "metadata", "flickr'd", "flickr", "latitude",
             "longitude", "nikon")



# Read GPS coordinates and date from EXIF-record of a .jpg file
# Gene Cash's EXIF.py library 1.1.1 is used
def getEXIF(filename):
      lat = 1
      lon = 1
      date = 1
      file = open(filename)
      exif = EXIF.process_file(file, details = False)
      for e in exif.keys():

             # Retrieve GPS coordinates
             if e == "GPS GPSLatitude":
                    l = str(exif[e].values[2]).split("/")
                    # Support cases where coordinates are not fractions
                    if len(l) == 2:
                           l = (float(l[0]) / float(l[1]))/60.0
                    else:
                           l = float(l[0])/60.0
                    l = (l + float(str(exif[e].values[1]))) / 60.0
                    lat = lat * (int(str(exif[e].values[0])) + l)
             elif e == "GPS GPSLongitude":
                    l = str(exif[e].values[2]).split("/")
                    # Support cases where coordinates are not fractions
                    if len(l) == 2:
                            l = (float(l[0]) / float(l[1]))/60.0
                    else:
                            l = float(l[0])/60.0
                    l = (l + float(str(exif[e].values[1]))) / 60.0
                    lon = lon * (int(str(exif[e].values[0])) + l)

            # Change to negative numbers (if south or west)
            elif e == "GPS GPSLatitudeRef":
                    if str(exif[e].values[0]) == "S":
                            lat = lat * (-1)
            elif e == "GPS GPSLongitudeRef":
                    if str(exif[e].values[0]) == "W":
                            lon = lon * (-1)

            # Retrieve date
            elif e == "EXIF DateTimeOriginal":
                    date = str(exif[e].values)

        return lat, lon, date



# Returns a list of synonyms of "word" (including word itself)
# STANDS4 Web Services - Synonyms API v1 is used
def get_synonyms(word):
        url =   "http://www.abbreviations.com/services/v1/syno.aspx?tokenid=(removed)&word="
                + word.replace(" ", "%20")
        response = urllib.urlopen(url).read()
        lindex = response.find("<synonyms>")
        rindex = response.find("</synonyms>")
        list_of_synonyms = response[lindex+10:rindex].replace(",", " OR ")
        if list_of_synonyms and (list_of_synonyms != word):
                # Add the word itself also to the list of synonyms
                list_of_synonyms = word + " OR " + list_of_synonyms
        else:
                # Did not find any synonyms (use only “word”)
                list_of_synonyms = word
        return list_of_synonyms



# Return search parameters to use based on main category (cat)
def category_handler(cat, sub_cat, date):

        # The text to use when searching Flickr
        query_text = ""

        if (cat == "object") or (cat == "place"):

                # Use images from all dates
            min_taken_date = "1900-01-01 00:00:00";
            max_taken_date = "2020-06-30 23:59:59";

            # Use sub-category as search parameter
            # ... and use all synonyms of the sub-category
            query_text = get_synonyms(sub_cat)

    elif cat.split("_")[0] == "event":

            # Initialize date and time variables
            d = date.split(" ")[0].split(":")
            d = datetime.date(int(d[0]), int(d[1]), int(d[2]))
            t = date.split(" ")[1].split(":")
            t = datetime.time(int(t[0]), int(t[1]), int(t[2]))
            date = datetime.datetime.combine(d, t)

            # Differentiate between short and long events
            if cat.split("_")[1] == "short":
                    days = 1                # The number of days to change the date with
                    #query_text = ""        # Do not use sub-category as search parameter

            elif cat.split("_")[1] == "long":

                    days = 30               # The number of days to change the date with

                    # Use sub-category as search parameter
                    # ... and use all synonyms of the sub-category
                    query_text = get_synonyms(sub_cat)

            else:
                    raise "Fatal error, unknown category"

            # Make a time interval using "days" variable
            delta = datetime.timedelta(days)

            # Set dates based on time interval
            min_taken_date = str(date - delta)
            max_taken_date = str(date + delta)

    # Used for LoTagr
    elif cat == "simple":
            min_taken_date = "1900-01-01 00:00:00";
            max_taken_date = "2020-06-30 23:59:59";
            query_text = ""

    else:
            raise "Fatal error, unknown category"

    return query_text, min_taken_date, max_taken_date



# Search Flickr for images and store their tags
# Returns a list of tags and amount of images used
def search(flickr, lat, lon, query_text, min_taken_date, max_taken_date):
```

```
tags = []               # The output (tags and their frequency)
id_list = []            # List of image id's (used to ensure that each image is processed only once)
user_list = []          # List of user id's that have at least one image in result set
users_tag_list = []     # List of users and their tags
                        # (used to prevent usage of same tag more than once per user)

images_used = 0         # Number of used images = size of result set
total_images_used = 0 # Total number of images (including several per user and those with no tags)
page_nr = 1             # The page number of the result page to start the search with
radius = START_RADIUS # The radius to use in the search

# The search process will continue until this number is reached
required_images = INITIAL_REQ_IMAGES

# Search while not enough images in result set
while images_used < required_images:

        # Search for images on flickr
        try:
                new_set =       flickr.photos_search(extras="tags", page=page_nr, per_page=250,
                                text=query_text, min_taken_date=min_taken_date,
                                max_taken_date=max_taken_date, lat=lat, lon=lon, radius=radius)
        except:
                # Operation timed out, try again (happens occasionally...)
                try:
                        new_set =       flickr.photos_search(extras="tags", page=page_nr, per_page=250,
                                        text=query_text, min_taken_date=min_taken_date,
                                        max_taken_date=max_taken_date, lat=lat, lon=lon, radius=radius)

                except:
                        raise "Problem with Flickr, please try again..."

        # Locate image set
        new_set = new_set.getchildren()[0].getchildren()

        # For each image
        for image in new_set:

                # Ensure that same image is not used more than once
                image_id = image.get("id")
                if image_id not in id_list:
                        id_list.append(image_id)

                        # Collect list of tags of this image
                        list_of_tags = image.get("tags")

                        # Encode all characters to ascii
                        list_of_tags =        unicodedata.normalize('NFKD', unicode(list_of_tags))
                                              .encode('ascii','ignore')

                        # Add user to user list
                        user = (image.get("owner"))
                        if user not in user_list:
```

```
            user_list.append(user)
            users_tag_list.append((user, []))

            # Increase images used
            # (but only if more than 1 tag is available)
            if len(list_of_tags.split(" ")) > 1:
                images_used += 1

        # Process each tag in list of tags
        for tag in list_of_tags.split(" "):

            # Check if the tag already exists in result set
            tag_exist = False # Assume it does not exist initially

            for k in tags:
                if k[1] == tag:

                    # The tag does exist in result set
                    tag_exist = True

                    # Only use tag if it is not added before from same user
                    for i in users_tag_list:
                        # Locate users tag list
                        if i[0] == user:
                            if tag not in i[1]:

                                # Add tag (increase frequency)
                                tags.remove((k[0], tag))
                                tags.append((k[0]+1, tag))

                                # Add tag to users tag list
                                i[1].append(tag)

                                # End user search
                                break

                    # End processing of current tag
                    break

            # Add tag with frequency 1 if it does not already exist in result set
            if not tag_exist:
                tags.append((1, tag))

        # Increse total images used (only for statistics)
        total_images_used += 1

        # End image search process if enough images are processed
        if images_used >= required_images:
            break

# Go to next result page (if possible)
if len(new_set) == 250:
    page_nr += 1
# Else do a new search with increased radius
else:
```

```
                # Increase radius
                radius = radius * RADIUS_MULTIPLIER

                # Lower required images
                required_images = required_images - (required_images/LIMIT_FACTOR)

                # New search will be performed, so begin at first result page
                page_nr = 1

            # Change radius to whole number for convenience
            if radius == 1.024:
                radius = 1

            # Ensure that maximum radius is not exceeded
            if radius > MAX_FLICKR_RADIUS:
                required_images = 0

        print  "Used " + str(images_used) + " (" + str(total_images_used)+ ") images with radius at "
            + str(radius/RADIUS_MULTIPLIER) + " km"

        # Return list of tags and number of images used
        return tags, images_used



    # Filter out tags from the tag filter and tags with low frequency
    def tag_filtering(tags, nr_of_images):

        # Tags with lower frequency than min_freq will be filtered
        min_freq = nr_of_images * FREQUENCY_LIMIT

        # Filter tags
        for i in list(tags):
            if i[1] in TAG_FILTER:
                tags.remove(i)
            elif i[0] <= min_freq:
                tags.remove(i)
        return tags



    # Handle whitespaces
    def handle_whitespaces(tags):

        # Search for duplicates like "Big Ben" and "BigBen"
        for i in tags:
            for j in tags:
                if j[1] != i[1]:
                    if (j[1].replace(" ", "") == i[1]) | (i[1].replace(" ", "") == j[1]):

                        # Remove the two duplicate tags from the list
                        tags.remove(i)
                        tags.remove(j)
```

```
                                                # Add highest frequent tag with updated frequency
                                                if i[0] >= j[0]:
                                                        tags.append((i[0]+j[0], i[1]))
                                                else:
                                                        tags.append((i[0]+j[0], j[1]))

                                                # Break inner for loop (go to next tag)
                                                break
        return tag



# Handle upper and lower cases
def handle_upper_and_lower_cases(tags):

        # Search for duplicates like "Big Ben" and "big ben"
        for i in tags:
                for j in tags:
                        if j[1] != i[1]:
                                if j[1].lower() == i[1].lower():

                                        # Remove the two duplicate tags from the list
                                        tags.remove(i)
                                        tags.remove(j)

                                        # Add highest frequent tag with updated frequency
                                        if i[0] >= j[0]:
                                                tags.append((i[0]+j[0], i[1]))
                                        else:
                                                tags.append((i[0]+j[0], j[1]))

                                        # Break inner for loop
                                        break
        return tags


# Connect to FlickrAPI using my api_key and api_secret
def connect2flickrapi():
        api_key = "(removed)"
        api_secret = "(removed)"
        return flickrapi.FlickrAPI(api_key, api_secret)


# Save the tags to disk
def store(tags, main, sub, id):
        file = open("output/" + main + "__" + id + "__" + sub + ".txt", "w")
        for i in tags:
                file.write(str(i[0]) + ":::" + str(i[1]) + "\n")
        if len(tags) == 0:
                file.write("no tags found:::0\n")
        file.close()


# SimpleTagr
# (One of the comparison systems)
def simple_tagr(lat, lon):
	flickr = connect2flickrapi()
	tags = []
	new_set =	flickr.photos_search(extras="tags", radius = 10, lat=lat, lon=lon, min_taken_date =
			"1900-01-01 00:00:00", max_taken_date = "2020-06-30 23:59:59")
	new_set = new_set.getchildren()[0].getchildren()
	for image in new_set:
		list_of_tags = image.get("tags").lower()
		for tag in list_of_tags.split(" "):
			tag = unicodedata.normalize('NFKD', unicode(tag)).encode('ascii','ignore')
			tag_exist = False	# Does the tag already exist in the list?
			for k in tags:
				# Check if tag already is in the list
				if k[1] == tag:
					tag_exist = True # Mark that the tag exists in the list

					# Increase frequency of tag
					tags.remove((k[0], tag))
					tags.append((k[0]+1, tag))

					break # End processing of current tag (go to next tag)

			# Add the tag with frequency 1 if it does not exist from before
			if not tag_exist:
				tags.append((1, tag))

	# Sort and filter out tags with low frequencies
	tags.sort(reverse=True)
	for i in list(tags):
		if i[0] <= (len(new_set) * FREQUENCY LIMIT):
			tags.remove(i)

	# Save results to file
	file = open("output/simpletagr.txt", "w")
	for i in tags:
		file.write(str(i[0]) + ":::" + str(i[1]) + "\n")
	file.close()


# The main function
if __name__ == "__main__":

	# Store start time
	time_taken = time.time()

	# Retrieve GPS coordinates and date
	if sys.argv[1] == "noimage":
		# Date, lat and lon passed directly as parameters
		lat = sys.argv[4]
		lon = sys.argv[5]
		date = sys.argv[6] + " 12:00:00"
```

```
elif sys.argv[1] == "image":

        # Add upload folder to specified image
        query_image = "upload/" + sys.argv[4].lower()

        # Read EXIF-record of query image
        lat, lon, date = getEXIF(query_image)
        if 1 in (lat, lon, date):
                print "Error reading EXIF-record..."
                print "Are you sure GPS coordinates and date is available in EXIF-header?"
                sys.exit()

# Perform the simple comparison system (SimpleTagr)
elif sys.argv[1] == "simpletagr":
        if sys.argv[2].split(".")[1].isdigit():
                lat = sys.argv[2]
                lon = sys.argv[3]
        else:
                lat, lon, date = getEXIF("upload/" + sys.argv[2].lower())
        simple_tagr(lat, lon)
        print "SimpleTagr used " + str(round((time.time() - time_taken), 1)) + " seconds."
        sys.exit() # No need to do more
else:
        print "Fatal error, wrong parameters"
        sys.exit()

# Read category parameters
main_cat = sys.argv[2].lower()
sub_cat = sys.argv[3].lower()

# Connect to flickrapi
flickr = connect2flickrapi()

# Start the category handler
query_text, min_taken_date, max_taken_date = category_handler(main_cat, sub_cat, date)

# Start the search process
tags, images_used = search(flickr, lat, lon, query_text, min_taken_date, max_taken_date)

# Tag processing
tags = handle_whitespaces(tags)                 # Handle whitespaces
tags = handle_upper_and_lower_cases(tags)       # Handle upper/lower cases
tags = tag_filtering(tags, images_used)         # Tag filtering
tags.sort(reverse=True)                         # Sort tags

# Store list of tags to disk
store(tags, main_cat, sub_cat, sys.argv[4])

# Calculate and output time taken
time_taken = round((time.time() - time_taken), 1)
print " and it took " + str(time_taken) + " seconds."
```

Appendix C – Web interface code

The web interface code consists of 10 files:

1. index.html
2. menu.html
3. useexample.pl
4. usequery.pl
5. upload.pl
6. manually.pl
7. runscript.pl
8. example_images.pl
9. newcat.pl
10. makecat.pl

C.1 index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>

<head> <meta http-equiv="Content-Type" content="text/html; charset=utf-
8"></meta><title>LoCaTagr</title> </head>

<body bgcolor="#2786C4">

<center> <h1>LoCaTagr</h1> </center>

<div style="background:transparent; border-top:5px solid #000000; width:1000px;"></div>

<div style="position: absolute; width: 164px; height: 538px; z-index: 1; left: 12px; top:
130px" id="layer1">



<table style="border-collapse: collapse" width="200" border="3" bordercolor=black cellpad-
ding="7" cellspacing="0">
  <tr>
       <td bgcolor="#2377AF">
               <center><a href='index.html'style='text-decoration:none'><font col-
or="#000000">Home</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='useexample.pl'style='text-decoration:none'><font color="#000000">Use
example image</font></a></center></td>
```

```
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='usequery.pl'style='text-decoration:none'><font color="#000000">Use
with query image</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='manually.pl'style='text-decoration:none'><font color="#000000">Use
without image</font></a></center></td>
  </tr>
  <tr>
       <td bgcolor="#2377AF">
        <center><a href='example_images.pl'style='text-decoration:none'><font col-
or="#000000">Download example images</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='newcat.pl'style='text-decoration:none'><font color="#000000">Make a
new category</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='index.html'style='text-decoration:none'><font col-
or="#000000">Help</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='index.html'style='text-decoration:none'><font col-
or="#000000">About</font></a></center></td>
  </tr>
</table>
</div>

<div style="position: absolute; width: 896px; height: 538px; z-index: 2; left: 250px; top:
130px" id="layer2">


<font face="Arial">
<p><b>
LoCaTagr is an automatic image tagging system using location, category<br>
and date to find a set of (hopefully) relevant tags for your image.
</b></p>

<p><b>
You need to provide a geo-referenced image and an image category.<br>
New image categories can be made if the image does not fit into one <br>
of the existing categories.
</b></p>

<p><b>
You can either<br>
 - use one of the provided example images<br>
 - use one of your own images (must have have GPS and date in its EXIF header)<br>
 - plot GPS coordinates and dates manually without using any image<br>
 - download the provided example images and use them as query images (i.e. the<br>
   same as using one of the provided example images)
</b></p>

<p><b>
LoCaTagr use images on Flickr as basis for tagging the query image.  <br>
Therefore, there must be enough images available on Flickr that is relevant  <br>
for your image before LoCaTagr is able to tag your image with relevant tags.
</b></p>

<p><b>
Regarding the performance, it is important to remember that this website is<br>
currently a prototype running LoCaTagr and the two comparison systems LoTagr <br>
and SimpleTagr. The runtime can therefore seem long as it can take around a <br>
minute to compute results for all the three systems. Further, as mentioned <br>
in the implementation, LoCaTagr is not optimized with regards to performance.
</b></p>

<p><b>
This site works best with Firefox. <br>
It is also known to work with Google Chrome, Safari and Internet Explorer.<br>
(Opera is currently not supported).
</b></p>

<p><b>
The project is made as part of a master thesis in Computer Science.<br>
Master thesis: <a href="http://caim00.cs.uit.no/locatagr/master.pdf">Automatic Image Tagging
based on Context Information</a>.<br>
The work is part of the <a href="http://caim.uib.no/">CAIM project</a>.<br>
Made by Martin HÃ¦tta Evertsen<br>
Contact: mhe023@post.uit.no<br>
<br>
<br>
</b></p>
</font>

</div>
</body>
</html>
```

C.2 menu.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>

<head> <meta http-equiv="Content-Type" content="text/html; charset=utf-
8"></meta><title>LoCaTagr</title> </head>

<body bgcolor="#2786C4">

<center> <h1>LoCaTagr</h1> </center>

<div style="background:transparent; border-top:5px solid #000000; width:1000px;"></div>

<div style="position: absolute; width: 164px; height: 538px; z-index: 1; left: 12px; top:
130px" id="layer1">



<table style="border-collapse: collapse" width="200" border="3" bordercolor=black cellpad-
ding="7" cellspacing="0">
  <tr>
       <td bgcolor="#2377AF">
              <center><a href='index.html'style='text-decoration:none'><font col-
or="#000000">Home</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='useexample.pl'style='text-decoration:none'><font color="#000000">Use
example image</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='usequery.pl'style='text-decoration:none'><font color="#000000">Use
with query image</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='manually.pl'style='text-decoration:none'><font color="#000000">Use
without image</font></a></center></td>
  </tr>
  <tr>
       <td bgcolor="#2377AF">
        <center><a href='example_images.pl'style='text-decoration:none'><font col-
or="#000000">Download example images</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='newcat.pl'style='text-decoration:none'><font color="#000000">Make a
new category</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='index.html'style='text-decoration:none'><font col-
or="#000000">Help</font></a></center></td>
  </tr>
  <tr>
    <td bgcolor="#2377AF">
        <center><a href='index.html'style='text-decoration:none'><font col-
or="#000000">About</font></a></center></td>
  </tr>
</table>
</div>

<div style="position: absolute; width: 896px; height: 538px; z-index: 2; left: 250px; top:
130px" id="layer2">
```

C.3 useexample.pl

```perl
#!c:/perl/bin/perl.exe
use CGI;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
       print $line;
}

# Make form displaying example images
$html = "
   <b><p>Please choose one of the provided example images</p> </b>

  <form action='upload.pl' method='post'
       enctype='multipart/form-data'>

       <p> <select name = 'photo'>
";
print $html;

@files = <verified_examples/*.jpg>;
foreach $file (@files) {
       @names = split("/", $file);
       $name = @names[1];
       print "<option value = $name>$name</option>";
}

       print '<p><input type="hidden" name="do_upload" value="no"/></p>';
       print "<input type='submit' name='Submit' value='Select' />";
       print "</form>";

print "</div>";
print "</body>";
print "</html>";
```

C.4 usequery.pl

```
#!c:/perl/bin/perl.exe
use CGI;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
        print $line;
}

# Make form for specifying query image
$html = '

<b>Please specify your query image...</b>

<form action="upload.pl" method="post"
enctype="multipart/form-data">
     <p><input type="file" name="photo" /></p>

                <p><input type="submit" name="Submit" value="Upload" /></p>
</form>

</div>
</body>
</html>

';
print $html;
```

C.5 upload.pl

```
#!c:/perl/bin/perl.exe
use CGI;
use File::Basename;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
        print $line;
}

# Set and retrieve variables
$CGI::POST_MAX = 1024 * 5000;
$safe_filename_characters = "a-zA-Z0-9_.-";
$upload_dir = "upload";
$query = new CGI;
$filename = $query->param("photo");
$do_upload = $query->param("do_upload");

# Special case when using example images
if ($do_upload eq "no") {
        $folder = "verified_examples";
}

# Regular file upload
else {
        if (!$filename) {
         print "There was a problem uploading your image (try a smaller file).";
         exit;
        }

        $folder = "upload";

        ($name, $path, $extension) = fileparse ( $filename, '\..*' );
        $filename = $name . $extension;
        $filename =~ tr/ /_/;  # Change whitespace to underline
        $filename =~ s/[^$safe_filename_characters]//g;

        if ($filename =~ /^([$safe_filename_characters]+)$/) {
                $filename = $1;
        } else {
                die "Filename contains invalid characters";
        }

        $upload_filehandle = $query->upload("photo");

        open ( UPLOADFILE, ">$upload_dir/$filename" ) or die "$!";
        binmode UPLOADFILE;
        while (<$upload_filehandle>) {
                print UPLOADFILE;
        }

        close UPLOADFILE;
        print "<b><p>Image successfully uploaded to server!</p> </b>";
}

# Make form for choosing category
$html = "
   <p><img src='$folder/$filename' width='250' heigth='250'  /></p>

  <form action='runscript.pl' method='post'
        enctype='multipart/form-data'>


         <b><p>Please specify the image category</p> </b>
         <p> <select name = 'sub'>


";
print $html;

# Read categories
open FILE, "categories.txt";
@lines = <FILE>;
close FILE;

@lines = sort @lines; # Sort categories

# Display categories
for $item (@lines) {
        @tuple = split(":::", $item);
        print "<option value = \"$tuple[0]\">$tuple[0]</option>";
}

$html =
"
        <input type='hidden' name='file' value=$filename>
        <input type='hidden' name='date' value='nodate'>
        <input type='submit' name='Submit' value='Find Tags!' />
        </form>

</div>
</body>
</html>
";
print $html;
```

C.6 runscript.pl

```
#!c:/perl/bin/perl.exe
use CGI;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
        print $line;
}

# Collect input parameters
$query = new CGI;
$image = $query->param("file");
$sub = $query->param("sub");
$lat = $query->param("lat");
$lon = $query->param("lon");
$date = $query->param("date");

# Retrieve main category using sub-category
open FILE, "categories.txt";
@lines = <FILE>;
close FILE;
for $item (@lines) {
        @tuple = split(":::", $item);
        if ($tuple[0] eq $sub) {
                $main = $tuple[1];
                chop($main);
        }
}

# Print debug info
print "<b>Processing the image...<br>";
print "(This can take several seconds, please wait...)</b>";
print "<br><br>";
print "Main category: $main";
print "<br>";
print "Sub category: $sub";
print "<br>";
print "Image: $image";
print "<br><br>";

# Start script on server ...
# ... either using query image
if ($date eq "nodate") {
        unlink("output/" . $main . "__" . $image . "__" . $sub .".txt");
        @ar = ('"LoCaTagr.py"', '"image"', '"' . $main . '"', '"' . $sub . '"', '"' . $image .
'"');
```

```
        $identifier = $image;
}
# ... or manually plotted GPS and date
else {
        unlink("output/" . $main . "__" . $lat . "__" . $sub .".txt");
        @ar = ('"LoCaTagr.py"', '"noimage"', '"' . $main . '"', '"' . $sub . '"', '"' . $lat .
'"', '"' . $lon . '"', '"' . $date . '"');
        $identifier = $lat;
}
system(@ar);

# Read results from disk
open FILE, "output/" . $main . "__" . $identifier . "__" . $sub . ".txt";
@lines = <FILE>;
close FILE;

# Dirty test to check if able to read from EXIF-header
if ($lines[0] eq "") {
        print "<br><b>Error, please try again!";
        print "<br>Make sure you have provided correct information</b>";

} else {

        # Display table with list of tags
        print "<br>";
        print "<br>";
        print "<b>Result:</b>";

        print '<table style="border-collapse: collapse" width="200" bordercolor="#000000" bor-
der="3" cellpadding="5" cellspacing="0">';

        print "<tr>";
        print "<td><b><center>Tag</center></b></td>";
        print "<td><b><center>Frequency</center></b></td>";
        print "</tr>";

        for $item (@lines) {
                @tuple = split(":::", $item);
                print "<tr>";
                print "<td><center>$tuple[1]</center></td>";
                print "<td><center>$tuple[0]</center></td>";
                print "</tr>";
        }

        print "</table>";


        # Make comparison table
        print "<br><br><br>Comparison table will appear below shortly, please be pa-
tient...<br>";
```

```
        # Collect comparison results (LoTagr)
        # ... either using query image ...
        if ($date eq "nodate") {
                @ar = ('"LoCaTagr.py"', '"image"', '"simple"', '"simple"', '"' . $image . '"');
                $identifier = $image;
        }
        # ... or manually plotted GPS and date
        else {
                @ar = ('"LoCaTagr.py"', '"noimage"', '"simple"', '"simple"', '"' . $lat . '"',
'"' . $lon . '"', '"' . $date . '"');
                $identifier = $lat;
        }
        system(@ar);


        # Collect comparison results (SimpleTagr)
        # ... either using query image ...
        if ($date eq "nodate") {
                @ar = ('"LoCaTagr.py"', '"simpletagr"', '"' . $image . '"');
        }
        # ... or manually plotted GPS and date
        else {
                @ar = ('"LoCaTagr.py"', '"simpletagr"', '"' . $lat . '"', '"' . $lon . '"');
        }
        system(@ar);


        print "<br><br>";
        print "<b>Comparison table</b>";

        print '<table style="border-collapse: collapse" width="200" bordercolor="#000000" bor-
der="3" cellpadding="5" cellspacing="0">';
        print '<tr valign="top">';
        print "<td>";


        # Print results for LoCaTagr
        print '<table style="border-collapse: collapse" width="200" bordercolor="#000000" bor-
der="3" cellpadding="5" cellspacing="0">';
        print "<caption>LoCaTagr</caption>";

        print "<tr>";
        print "<td><b><center>Tag</center></b></td>";
        print "<td><b><center>Frequency</center></b></td>";
        print "</tr>";

        for $item (@lines) {
                @tuple = split(":::", $item);
                print "<tr>";
                print "<td><center>$tuple[1]</center></td>";
                print "<td><center>$tuple[0]</center></td>";
                print "</tr>";
        }
        print "</table>";
        print "</td>";
        print "<td>";


        # Print result for LoTagr
        open FILE, "output/simple__" . $identifier . "__simple.txt";
        @lines = <FILE>;
        close FILE;

        print '<table style="border-collapse: collapse" width="200" bordercolor="#000000" bor-
der="3" cellpadding="5" cellspacing="0">';
        print "<caption>LoTagr</caption>";

        print "<tr>";
        print "<td><b><center>Tag</center></b></td>";
        print "<td><b><center>Frequency</center></b></td>";
        print "</tr>";

        for $item (@lines) {
                @tuple = split(":::", $item);
                print "<tr>";
                print "<td><center>$tuple[1]</center></td>";
                print "<td><center>$tuple[0]</center></td>";
                print "</tr>";
        }


        print "</table>";
        print "</td>";
        print "<td>";


        # Print results for SimpleTagr
        open FILE, "output/simpletagr.txt";
        @lines = <FILE>;
        close FILE;

        print '<table style="border-collapse: collapse" width="200" bordercolor="#000000" bor-
der="3" cellpadding="5" cellspacing="0">';
        print "<caption>SimpleTagr</caption>";
        print "<tr>";
        print "<td><b><center>Tag</center></b></td>";
        print "<td><b><center>Frequency</center></b></td>";
        print "</tr>";

        for $item (@lines) {
                @tuple = split(":::", $item);
                print "<tr>";
                print "<td><center>$tuple[1]</center></td>";
                print "<td><center>$tuple[0]</center></td>";
                print "</tr>";
        }

        print "</table>";
        print "</td>";
        print "</tr>";
        print "</table>";

} # end of else from return value

print "<br></div></body></html>\n";
```

C.7 manually.pl

```
#!c:/perl/bin/perl.exe
use CGI;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
        print $line;
}

# Make form allowing the user to plot GPS and date manually
my $html =
'
<b>Plot GPS coordinates and date manually</b>

<form action="runscript.pl" method="post"
enctype="multipart/form-data">

        <p>Latitude (DDD.DDDD): <input type="text" name="lat" /></p>
        <p>Longitude (DDD.DDDD): <input type="text" name="lon" /></p>
        <p>Date (YYYY:MM:DD): <input type="text" name="date" /></p>
        <p>Category:  <select name = "sub"> /></p>

';
print $html;

# Read and display categories
open FILE, "categories.txt";
@lines = <FILE>;
close FILE;
@lines = sort @lines; # Sort categories

for $item (@lines) {
        @tuple = split(":::", $item);
        print "<option value = \"$tuple[0]\">$tuple[0] </option>";
}

$html = '
        <p><input type="hidden" name="file" value="not using any image"/></p>
        <p><input type="submit" name="Submit" value="Find Tags!" /></p>
</form>

<b>
        <br>

        GPS coordinates must be in decimal degree form.<br>
        Click
        <a href="http://www.getlatlon.com/">here</a>
        to find GPS coordinates.
```

```
   </div>
   </body>
   </html>

';
print $html;
```

C.8 example_images.pl

```
#!c:/perl/bin/perl.exe
use CGI;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
        print $line;
}

# Print example images
print "<b><p>Example images: <br></p> </b>";

@files = <verified_examples/*.jpg>;
foreach $file (@files) {
        print "<p><img src='$file'  width='250' heigth='250'  /></p>";

        @tuple = split("/", $file);
        print "<p>$tuple[1]</p>";
        print "<br></br>";
        print "<br></br>";
}

print "</div></body></html>\n";
```

C.9 newcat.pl

```
#!c:/perl/bin/perl.exe
use CGI;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
       print $line;
}

# Make form for making new category
$html =
'
<b>Create new category</b>

<form action="makecat.pl" method="post"
enctype="multipart/form-data">
    <p>Category name: <input type="text" name="sub" /></p>

       <p>Main Category: <select name = "main">
       <option value = "object">Object</option>
       <option value = "place">Place</option>
       <option value = "event_short">Event (short-lasting, e.g. concert)</option>
       <option value = "event_long">Event (long-lasting, e.g. festival)</option>
       </select></p>
       <input type="submit" name="Submit" value="Make new category" />

</form>

';
print $html;


# Make form for deleting categories

# Read categories
open FILE, "categories.txt";
@lines = <FILE>;
close FILE;
@lines = sort @lines; # Sort categories

print "<br><br><br><br>";
print '<form action="makecat.pl" method="post"';
print 'enctype="multipart/form-data">';
print '<b>Delete existing category</b>';
print "<p> <select name = 'sub'>";

# Display categories
for $item (@lines) {
       @tuple = split(":::", $item);
```

```
        print "<option value = \"$tuple[0]\">$tuple[0]</option>";
}

        print "<input type='hidden' name='main' value='delete'>";
        print '<input type="submit" name="Submit" value="Delete category" />';

print '</form>';

print "</div></body></html>\n";
```

C.10 makecat.pl

```perl
#!c:/perl/bin/perl.exe
use CGI;
print "Content-Type: text/html\n\n";

# Make menu
open FILE, "menu.html";
@lines = <FILE>;
close FILE;
for $line (@lines) {
	print $line;
}

# Collect input
$query = new CGI;
$main = $query->param("main");
$sub = $query->param("sub");
$sub =~ s/^\s+//; # Remove leading spaces
$sub =~ s/\s+$//; # Remove trailing spaces
$sub =~ y/A-Z/a-z/; # Make all letters lowercase
$sub =~ s/\b(\w)/\U$1/g; # Capitalize

# Read existing categories
open FILE, "categories.txt";
@lines = <FILE>;
close FILE;

# Delete or create mode
if ($main eq "delete") {
	open FILE, ">categories.txt";
	for $item (@lines) {
		@tuple = split(":::", $item);
		if ($tuple[0] eq $sub) {
			# Do not insert it again
		}
		else {
			print FILE "$tuple[0]" . ":::" . "$tuple[1]";
		}
	}
	close FILE;
	print "<b>Category deleted!</b>";
}
else { # Create mode

	# Check if category already exist
	$already_exist = "False";
	for $item (@lines) {
		@tuple = split(":::", $item);
		if ($tuple[0] eq $sub) {
			$already_exist = "True";
```

```
                }
        }

        # Do not allow whitespace only
        if ($sub eq "") {
                print "<b>Category cannot consist of just whitespace!</b>";
        }

        # Add category if it does not exist from before
        elsif ($already_exist eq "False") {

                open (MYFILE, '>>categories.txt');
                print MYFILE "$sub" . ":::" . "$main\n";
                close (MYFILE);

                print "<b>Category made successfully!</b>";
                print "<br>";
                print "<b>You can not select your category from the list of categories.</b>";

        }
        else {
                print "<b>Category already exist!</b>";
        }
}

print "</div></body></html>\n";
```