Bachelor Thesis

Retrieval of Web Images for Computer Vision Research

September 28, 2009

Author:
Maikel Linke (NWI)
malinke@techfak.uni-bielefeld.de

Supervisors:
Dipl.-Inform. Marco Kortkamp
PD Dr.-Ing. Sven Wachsmuth
Abstract

Today, the web is an important source of image data for computer vision research. Search engines and online photo albums allow the retrieval of images related to a given keyword. This work contains an overview of these services and their usage. Additionally, it describes the development of a common programming interface to access different services. The developed Java library implements the most prominent services.
1 Introduction

Computer vision research requires large datasets of images. The World Wide Web is an immense source of data and rapidly growing\textsuperscript{1}. The number of images on websites is growing with the web. And photo sharing services like Flickr report thousands of uploads per minute\textsuperscript{2}. These images can be accessed through a keyword based search provided by different search engines and application programming interfaces (APIs).

Therefore a lot of research uses the web as an image data source. Chapter 2 gives an overview of some important papers. While older algorithms used some fixed datasets to classify visual objects, new ones try to learn from noisy datasets retrieved from image search engines or photo sharing services\textsuperscript{3}. Most approaches are tested with only a few services or just one. But the more sources an algorithm supports with—constant quality—the more general and adaptive it is. Some papers name the sources only as example to indicate, that their algorithm works with multiple sources. This is the first motivation to develop a common interface for different sources. So algorithms can be developed source-independently and their actual task can be focused.

Another motivation to combine multiple services is, that all services are limited. Most search engines limit the number of search results to one thousand per query. With multiple services more images can be retrieved. But there are also some techniques like translating the query to other languages to achieve more results. A common library could provide such techniques for all supported services.

Third, this work tries to expose further advantages of combining multiple services. Possible advantages are

- easy using through a common interface,
- efficiency by a common infrastructure like a thread pool and
- an easily extendable library to maximize the number of results and to keep up with the ever-changing web.

In the end, such an open library could support the community by merging web image retrieval knowledge. Future work in computer vision research could use such a library instead of reinventing the wheel every time.

Chapter 2 introduces literature concerned with datasets and web images. There is also evaluated, which service-support should be implemented first. The requirements of the system are explored in Chapter 3. This leads to the software design presented in

\textsuperscript{1}http://news.netcraft.com/archives/web_server_survey.html
\textsuperscript{2}http://www.flickr.com/
\textsuperscript{3}This evolution is described in the introduction of \cite{5}.
Chapter 4. In Chapter 5 is described, how the software works and how new modules are written to extend the library. Chapter 6 explains how to use the library as API user. An exemplary precision comparison of different services and a performance test of the developed library are described in Chapter 7. The results of this work are summarized in Chapter 8.
2 Related Work

The introduced works below describe computer vision algorithms. These algorithms need a large amount of image data. Focusing on the pure retrieval (collecting and downloading) of these images is a way of gaining hints how to design a web image retrieval system. For that purpose, this chapter is not sorted by papers. It is divided into the generation of datasets and the usage of search services. Both points to the ranking of these services to decide, which service-support should be implemented first.

2.1 Datasets generated from the Web

This section summarizes some popular datasets, which were build by using images from the web. Some datasets also contain non-web images. Retrieval details are not exposed in all cases. A statistical overview of the datasets is given by Table 2.1 on page 5.

**Caltech-101 (2004).** The paper [7] presents a way of learning object categories using prior information. The images were collected by using Google Images and then filtered manually. The result is a dataset of 101 object categories containing over 9,000 images. The dataset can be downloaded at [http://www.vision.caltech.edu/](http://www.vision.caltech.edu/).

**Faces in the Wild (2004).** In 2004 Berg et al. published [2]. It describes automatic association of faces and names from the caption of news images. For this purpose, they collected almost 45,000 images from Yahoo News. The dataset can be downloaded at [http://www.tamaraberg.com/faceDataset/](http://www.tamaraberg.com/faceDataset/).

**Pascal VOC (2005, 2006, ...).** The 2005 Pascal Visual Object Classes Challenge, described in [5], was a competition of twelve teams. The task was to recognize objects from four visual object classes. At a certain time a development kit and datasets were published. Three weeks later the test data has been released. One of the test datasets has been collected from Google Images and contained 654 images. In 2006 the VOC Challenge ([6]) used an image database collected from personal photographs, Flickr and the Microsoft Research Cambridge (MSRC). It contained over 5,000 images. The challenge still takes place every year and all datasets can be downloaded at [http://pascallin.ecs.soton.ac.uk/challenges/VOC/](http://pascallin.ecs.soton.ac.uk/challenges/VOC/).

**MSRC (2005).** Winn, Criminisi and Minka present an algorithm for automatic categorization of object classes from images in [27]. The test images have been collected from acquired photographs, images from the web and from a Pascal dataset
It is not mentioned, how the web images have been retrieved. The dataset can be downloaded at http://research.microsoft.com/en-us/projects/objectclassrecognition/.

**Animals on the Web (2006).** Berg and Forsyth searched for images using Google’s web search and combined text and visual information for clustering the downloaded images. They selected ten animals as categories and downloaded over 26,000 images. Afterwards they re-ranked the images to get good results. The category sizes and precision is described in [3]. The dataset can be downloaded at http://www.tamaraberg.com/animalDataset/.

**Caltech-256 (2007).** Very similar to the collecting approach of the Caltech-101 dataset is the collecting of the Caltech-256 dataset. In addition to Google Images also Picsearch has been used to collect the images. By combining the first results of two search engines the precision of the collected categories could be increased. Searching and Filtering resulted in 256 object categories with over 30,000 images. The dataset can be downloaded at http://www.vision.caltech.edu/.

**LabelMe (2008).** The goal of the work in [20] was to collect a large number of images with ground truth labels. Google Images, Flickr and Altavista Images have been used to collect the images and a new web-based tool to label the images was developed. In 2008 the dataset contained over 30,000 images with over 100,000 annotations in 183 categories. The dataset can be downloaded at http://labelme.csail.mit.edu/.

**ImageNet (2009).** [4] describes the building of ImageNet, a structured index like WordNet, but containing images. Several, not named search engines have been used to retrieve candidate images. The 5200 categories were filtered manually and build a dataset of 3.2 million images. The dataset can be downloaded at http://www.image-net.org/.

### 2.2 Image Services and their Usage

A lot of search services are provided in the web. Some of them are listed in this section. The list contains all services found in important papers plus randomly found services. An overview is given in Table 2.2.

**Altavista** provides its search service since 1995 and is one of the very first search engines, which is still on the market. It supports image, especially photo, and video search. Torralba et al. used Altavista, Ask, Flickr, Cydral, Google, Picsearch and Webshots to collect 80 million tiny images stored with 32 pixels height and width and analyzed object classification described in [22].
Table 2.1: Popular datasets, which were build using the web. Column four lists the number of annotations, while the annotation style is varying and not listed. This table does not claim to be complete.

Ask searches for websites, images, news, videos, maps and “answers”. No special options for the image search are provided, but there are distinct portals for some countries with different search results for the same keyword. Ask was used in [22] as described in the paragraph Altavista. Popescu, Moëllic and Millet selected Ask to build an image database of over 25,000 images because of the high precision for queries with unique terms. In [18] they wrote:

For a set of 20 queries with familiar concepts and 50 images per query, the mean precision is around 80%, while Picsearch, Yahoo! and Google obtain respectively 70%, 63% and 56%.

Bing is the name of Microsoft’s new search engine, online since June 2009. The previous version was called Live Search. Search results are accessible through an API supporting multiple protocols: JSON, XML and SOAP. During research for this work no paper using Bing was found. The likely reason is the new release of Bing. But also the old search engines of Microsoft could not be found in important papers.

Cydral is a pure image search engine used by [22] as mentioned in the paragraph Altavista. Until the middle of September 2009 it was not available. After that it provided only a few results1.

Flickr is a popular photo sharing service provided by Yahoo. The website reported, “5,831 uploads in the last minute”2. Multiple protocols are supported by the open API with the following response formats: REST, XML-RPC, SOAP, JSON, PHP and YQL.

---

142 images for “dog” and 91 images for “cat”, 09/15/2009
2http://www.flickr.com/, 09/08/2009, 16:41
In 2008 at least five important papers used Flickr: [15, 17, 19, 22, 28]. Li et al. generated 3D models of landmarks from a collection of landmark photos. Olivares, Ciaramita and van Zwol used annotated photos from Flickr to enhance keyword based searches by using Flickr notes to extract visual features. Raguram and Lazebnik used Flickr photos to find iconic images representing visual categories. The work of Torralba et al. is described in the paragraph Altavista. Yang, Chen and Wang took over two million tags from Flickr to build a demo version of “Web 2.0 Dictionary”. In 2009 [1] also presented a way of finding iconic images.

**Google Images** is separated from the paragraph Google Web, because it was used by the most papers focused in this research. The following 15 papers are using Google Images: [9] describes a visual category filter tested with Google Images. [26] groups Google’s results in clusters trained by human labeled images. [5] reports about the Pascal VOC 2005, where one test dataset was collected using Google Images. [8] presents an algorithm learning a category by its name without supervision. It retrieves the training data from Google Images. [12] groups the results of an image search engine and presents a new user interface to show and select the clustered search results (e.g. of Google Images). [13] clusters the results of six categories into groups of keywords. [25] studies the effectiveness of ontology based approaches in image retrieval. [10] describes the retrieval of the Caltech-256 dataset mentioned before. [21] uses textual and visual features to retrieve a large number of images for a given object class. The authors compare three approaches: First, collecting images from a website found with Google Web. Second, searching with Google Images and downloading all images from the websites of the found images. And third, downloading only the images found by Google Images. They measured the precision of all three approaches (same order): 26%, 4% and 39%. [14] merges results from Google Images and Google Web and re-ranks the images using the groundwork of [21]. [16] describes the re-ranking of Google’s image search results. A user selects wanted images and the algorithm ranks related images higher. [20] reports from the LabelMe dataset mentioned above. [22] uses Google Images among others and is described in the paragraph Altavista. [23] follows a multi-modality ontology approach to re-rank images. [24] constructs a multi-modality ontology from Wikipedia and combines this with the previous work. Google provides no API for its image search and limits the search results to 1000.

**Google Web** can also be used to search images. Most found websites contain images. Some are related to the keyword, although the precision is very low. The text of the website can be analyzed to evaluate the images. The previously mentioned [3] describes the search for animal images on websites returned by Google. Even in [21] this method is compared with Google’s image search and an approach between. [14] combines these two approaches.

**Picasa Web Albums** is a photo sharing service like Flickr. It is provided by Google and can be accessed through the Google Data API. [17, 28] mention this portal as alternative to Flickr, but do not use it.
**Picsearch** searches for images, but also supports web search. It can filter animations, colored pictures, portraits and some picture sizes. In [12] candidate images for clustering are retrieved from an image search engine (e.g. Picsearch). [10] reports from the Caltech-256 dataset generated with the results of Google Images and Picsearch. [22] is mentioned in *Altavista* and used Picsearch among many others.

**Webshots** is a photo sharing portal also used by [22].

**Yahoo** provides similar search services to Google. Many APIs are usable. The image search returns many Flickr photos\(^3\). Yahoo News provides a photo portal with photo captions used by [2, 11]. The aim of Berg et al. ([2]) was to identify the persons in the photos. Jeon and Manmatha ([11]) tried to label the photos automatically.

<table>
<thead>
<tr>
<th>Name</th>
<th>Interface(s)</th>
<th>Max. results</th>
<th>Used By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altavista</td>
<td></td>
<td>1020</td>
<td>[22]</td>
</tr>
<tr>
<td>Ask</td>
<td></td>
<td>320</td>
<td>[18, 22]</td>
</tr>
<tr>
<td>Bing</td>
<td>JSON, XML, SOAP</td>
<td>1000</td>
<td>[22]</td>
</tr>
<tr>
<td>Cydral</td>
<td></td>
<td>—</td>
<td>[22]</td>
</tr>
<tr>
<td>Flickr</td>
<td>REST, XML-RPC, SOAP, JSON, PHP, YQL</td>
<td>∞</td>
<td>[15, 17, 19, 22, 28, 1]</td>
</tr>
<tr>
<td>Google Images</td>
<td></td>
<td>1000</td>
<td>[9, 26, 5, 8, 12, 13, 25, 10, 21, 14, 16, 20, 22, 23, 24]</td>
</tr>
<tr>
<td>Google Web</td>
<td></td>
<td>1000</td>
<td>[3, 21, 14]</td>
</tr>
<tr>
<td>Picasa Web</td>
<td>Atom, JSON</td>
<td>∞</td>
<td>[17, 28]</td>
</tr>
<tr>
<td>Albums</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Picsearch</td>
<td></td>
<td>10000</td>
<td>[12, 10, 22]</td>
</tr>
<tr>
<td>Webshots</td>
<td></td>
<td>∞</td>
<td>[22]</td>
</tr>
<tr>
<td>Yahoo</td>
<td></td>
<td>1008</td>
<td>[2, 11] (News Photos)</td>
</tr>
</tbody>
</table>

Table 2.2: Popular image services. This table does not claim to be complete.

### 2.3 Ranking of Services

Not all services can be implemented in this work. Thus the services have to be ranked to decide, which service-support should be implemented first. Many criteria are possible: used by many papers, new and not studied, good API support, many results, good results, good performance, et cetera.

In this work the most crucial criterion is the number of papers, which had used the service. But the others also play a role. Additionally diverse services should be used to develop a library supporting most services. The following shows an ordered list of services with keyword comments, which criteria were most decisive.

\(^3\)Example: 11 of the first 18 results of a “dog” search came from Flickr, 09/10/2009, 10:15
1. Google Images, most used
2. Flickr, very often used, ready Java API
3. Bing, not used, but promising\textsuperscript{4} and API differing from Google Images and Flickr
4. Picsearch, often used, high number of results
5. Picasa Web Albums, used, Java API
6. Webshots, many results
7. Altavista
8. Ask, few results, but multiple portals with different results
9. Yahoo, overlap with Flickr
10. Google Web, web search not focused, low precision
11. Cydral, not available or very few results at the moment

The first three services have been implemented in the developed library. The remaining list can be seen as suggestion, which services should be supported next.

\textsuperscript{4}The evaluation described in Section 7.2 educes a relative low precision.
3 Requirements

The requirements of a web image retrieval library are evolved in this chapter. First, the domain of web image retrieval is introduced. Then Section 3.2 lists the goals of a library user, while Section 3.3 names the ones of a developer. From these goals further requirements are derived in Section 3.4.

3.1 The Domain of Web Image Retrieval

Most of the terms used in this domain were already mentioned in the previous chapters. Figure 3.1 connects these terms. Search is the main term. What the program should do, is searching for images and providing them. The Search provides a number of Web Images. The Query defines, what kind of images are requested. In this case: all images related to a given Keyword\(^1\). A Keyword comes from a Language, even if some words are the same in different languages. How many images are demanded, defines the Image Quantity. More parameters are imaginable\(^2\).

![Diagram of domain model without software specific terms.](image)

Figure 3.1: A domain model without software specific terms.

\(^1\)“...keyword-based search is the de-facto model for query formulation on the Web.” [17]

\(^2\)Common query parameters of the used services Bing, Flickr and Google are: activating a safe-search filter, setting up a license of the images and defining a media-type, for example “photos”. Each service has many more individual search parameters.
3.2 The User’s Goals

What are the user’s goals? He wants to retrieve and process images. What kind of processing he does, should not matter. To retrieve images, he needs something providing these images. This implicates two new terms—Web Image Receiver and Web Image Provider—generalizing the problem. The user also wants to

- use the library out of the box,
- learn the usage of the library quickly,
- define the image quantity,
- define how many images are needed,
- retrieve no duplicates, if not desired,
- know everything about found images,
- process the images efficiently, and
- select which services should be used.

3.3 The Developer’s Goals

The developer extends the library to increase the number of supported services. She wants to

- understand the library quickly,
- write as little code as possible,
- put aside common tasks, that are not service specific, and
- benefit from the already coded infrastructure.

3.4 Derived Requirements

What else implicate these goals? Some of them can be directly translated into software design. Others are more general.

To use the library out of the box, it needs a good default configuration. This is not possible in all cases. For example the Flickr API needs registered user keys for authentication. So this service should not be activated until these keys are given. This example also shows, that the services have to be initialized individually. So the library needs something managing these initializations, for example a factory\(^3\). The library

\(^3\)Explained in the Factory design pattern.
should also provide a minimal implementation for direct use. Such an implementation can also serve as an example code.

To learn the usage of the library quickly, it should provide a minimal number of public interfaces to use. The details should be encapsulated\(^4\).

Sometimes the quantity of available images for a given query is lower than the desired quantity. A very special query can result in no or only a few images. Therefore only a maximum of images is defined in a query.

Avoiding duplicate images needs a central register of all found images. The usage of this register should be optional.

To process the images efficiently, every image should be processed as soon as it is available. A concurrent search for more images should go on. This implicates a kind of thread management and an event-based data processing. This means, that the images are not returned as collection, but a \texttt{process image} method is called, when an image is available. The event-based processing is not constrained, but a simple solution for this problem.

On the other hand, the developer has her own goals. A simple architecture with clear programming interfaces are also needed. But the developer’s interfaces differ from the user’s interfaces. Therefore these interfaces should be divided into separate packages.

To write as little code as possible common abstract classes and tools should be given. That way the developer has to write only the service specific code. These tools should provide all common tasks and encapsulate details to allow the developer to put aside these details. The more reusable the code is designed, the more a developer can benefit from it.

All these requirements have been evolved step by step during the whole software development phase. They are included in the software design presented in the next chapter.

\(^4\)Encapsulation is an object oriented design principle, which hides complexity.
4 Design

In the previous chapter the requirements of a web image retrieval library are described. These lead to the design demonstrated in this chapter. It is divided into the user interfaces derived from the user’s perspective, and the remaining architecture behind these interfaces, which deals with the developer’s needs and all other derived requirements.

4.1 The User Interfaces

Derived from the user’s goals, a minimal set of public interfaces can be designed. It is presented in Figure 4.1. The client code can get WebImage Providers from the WebImageProviderFactory. Providers can be configured with query information and then be executed. Providing images runs concurrently to the rest of the client code. A found image is delivered through the processWebImage method of the WebImageReceiver interface. The client code has to implement this interface to receive the images.

![Image of user interfaces](image)

Figure 4.1: The basic set of user interfaces.

4.2 The Architecture Behind

Without further requirements, the whole rest of the code could be placed in the same package, not visible for the client code. But the developer needs some interfaces, too. So a package hierarchy is useful, where one package contains the user interfaces and one the developer interfaces. Additionally each implemented service (WebImageProvider) should gain its own package, because they will change independently, when a service changes. And implementations added by different developers are desirable.

13
This is the root package of the library.

The place for user interfaces.

The place for developer tools.

The places for each WebImageProvider.

Figure 4.2: Overview of all packages.

Figure 4.3: All classes in the provider package.
While the root package contains all user interfaces, the sub-package `provider` was created to store the developer tools. The first helpful class here is an `AbstractImageProvider`. It implements the `WebImageProvider` interface and carries for all architecture-implicated responsibilities: managing receivers, storing query configuration and handling of exceptions. The number of methods, that have to be implemented, shrinks from six to two.

The next responsibility an `WebImageProvider` has, is to run in a thread. This provides the `ThreadedImageProvider`. It extends the `AbstractImageProvider` and starts an own thread. The `providing finished` message is sent last in the thread. Only one abstract method has to be implemented.

Then, all implemented providers query an internet server to get image URLs for a given query. After that, they have to download the images and send them to all listening receivers. Collecting URLs, downloading images and calling the receivers are common tasks. But the collecting of URLs differs. The next abstraction is the class `DownloadingProvider`. All it needs is an `ImageUrlProvider`, which is gained via an abstract method.

These three abstract implementations cover all needed levels so far and all normal providers extend the `DownloadingProvider`. All other classes in this package are tools for common problems and can be used for own code. The `ReceiverCollection` can be used to store and inform receivers. To bind a `ReceiverCollection` to a provider, the `ReceiverCaller` can be used. An `ImageDownloader` requires such a caller for downloaded images and an `ImageUrlProvider` to get the image URLs. To download the adequate number of images it also needs an `ImageCounter`. While the bytes of an image are being downloaded, additional information, for example the URL, has to be stored. For this purpose the `FutureWebImage` stores these information. It also implements the `ByteArrayProcessor` interface to receive the downloaded bytes and send the ready web image via a `ReceiverCaller`.

One of the most important utility classes is the `ScheduledDownloader`. It manages a thread pool and is used for all downloads. One web host is not stressed by more than one download at a time. A delay can be given to make pauses between downloads from one host. This prevents blocking of requests. Via the `isNewUrl` method duplicate downloads can be avoided.

The `HtmlDecoder` replaces encoded ampersands in found URL strings. And the `GoogleTranslator` can translate text via Google’s translation service.

Two classes in the root package were not mentioned before. First, the `FileWritingReceiver` is an example implementation of `WebImageReceiver` and simply writes all received images to a directory. Second, the `LanguageIterator` generates providers for different languages and calls them with translated search keys.

Figure 4.2 presents all classes of the library at once. Package private classes are also visible. Generalizations and aggregations were all drawn, but not all simple usage-dependencies.

---

1There are two special providers in the root package. The `LanguageIterator` and the `AllInOneProvider` are decorating (design pattern) the providers. For this purpose, they only implement the `WebImageProvider` interface.
Figure 4.4: All classes sorted in packages. Not all dependencies are visible.
5 The Developer’s Guide

A developer wants to extend the library. In this case, she wants to write new implementations of WebImageProvider. There are two ways to realize this:

1. Modifying the existent source code.
2. Using the existent code as library and writing only new code.

The first manner is easier to code and described in the next two sections, while the second manner is better to maintain, but has some restrictions. These restrictions and some possible solutions are given in Section 5.3.

5.1 Implementing a new Provider

As seen in Section 4.2, the easiest way to implement a new image provider is to extend the DownloadingProvider. It asks for an ImageUrlProvider, which has to be given by implementing the createUrlProvider method. This URL provider only has to provide collections of image URLs. Downloading the images, delivering them to the receivers, counting the images and finishing the providing is done by the DownloadingProvider. Helpful tools are the ScheduledDownloader to download anything from a given URL and the ImageCounter to avoid providing too many URLs, which will not be used. The counter should be taken from the DownloadingProvider, since it uses the same instance to count the images.

If this solution does not fit, the ThreadedImageProvider can be used. An implementation has to fill the executeInThread method. The whole configuration of the provider can be accessed via get-methods of the extended classes. The finish signal is sent after returning of the executeInThread method. The responsibilities of the implementing class are:

- downloading of images,
- counting of images and not downloading more than the maximum,
- delivering of downloaded images,
- returning, when all downloads are finished.

Perhaps the ImageDownloader can be used to fulfill these tasks. Useful, in most cases, will be the ScheduledDownloader and the ReceiverCaller, which can be accessed via the ThreadedImageProvider. The FutureWebImage can be useful as well.
The simplest abstract implementation of WebImageProvider is the AbstractImageProvider. It only checks the right usage (state) of the provider, checks the given input and stores it.

5.2 Modifying existent Source Code

If a new provider has been implemented, it should be integrated in the user’s interfaces by the following steps:

- Adding an identifier of the new service to the WebImageService enumeration.
- Giving the ability to create new instances of the new provider to the WebImageProviderFactory.
- Eventually modifying the used configuration file to setup the factory correctly.

After these steps the new provider can be used like any other provider.

5.3 Limits of Pure Extending

If there is only new code to add, a new provider can be implemented as described above. What cannot be done is:

1. modifying WebImageService and
2. modifying WebImageProviderFactory.

WebImageService is a Java Enum, which cannot be extended. As a consequence, the new provider cannot be identified by an instance of WebImageService and not be recognized by an extended factory. If the client code depends on only one factory, an extended factory can be written, that takes general Enums as argument and evaluates, if it can handle this Enum as service.

It is also not possible to extend the Language enumeration. The supported languages and the translator tool have to be maintained in the library’s source code.

Last, the information provided by WebImage could be not sufficient. If more special information about web images is needed, there are two possible solutions:

1. Extending of WebImage. A special provider creates this type and received WebImages are casted, if they come from this provider.
2. Another special provider stores additional information and provides this via get-methods for given web images.

In the future, a solution should be found for the last problem. The remaining restrictions are bearable for the present and can be solved by adding new interfaces.
6 A brief API User Guide

The usage of the library follows three steps:

1. getting one or more web image providers,
2. configuring and executing,
3. receiving of web images.

These three steps are described in the next sections. First test code is printed in Listing 6.1. It is the shortest way to download some images and write them to the system’s temporary directory.

```java
/* setting variables */
final String searchKey = "dog";
final int numberOfImages = 3;
final WebImageReceiver receiver = new FileWritingReceiver();
/* get a special provider, 
   decorating all available providers */
final WebImageProvider provider = new AllInOneProvider();
/* configure and execute */
provider.addReceiver(receiver);
provider.configure(searchKey, numberOfImages);
provider.execute();
/* now receiving ... */
provider.waitForEndOfProviding();
```

Listing 6.1: A short example to use the library.

6.1 Getting Web Image Providers

The example above shows the easiest way of getting a provider. It creates an AllInOneProvider, which takes all available services and creates a provider of each service. The available services are provided by the WebImageProviderFactory class. Which services are available depends on which services are configured. All services can be configured via the setup method of the factory class. The configuration file could look like Listing 6.2 on the next page.
A more conventional way to get providers, is to query the WebImageProviderFactory. The `createWebImageProvider` method needs a service as argument and returns a new instance of the wanted provider type. Similar to this method works the `createProviderDistributor` method. It returns an instance of the `ProviderDistributor` interface, that is able to produce any number of providers of the previous given type, without demanding any parameter. This interface allow other classes to work service-independently. For example the `LanguageIterator` needs such a distributor to generate a provider for each language and configures them with translated search keys. The `AllInOneProvider` provides also a distributor for itself. This way it can be combined with the `LanguageIterator` to retrieve the maximum number of images.

### 6.2 Configuring and Executing

Each provider has four methods to be configured:

- `configure`
- `addReceiver`
- `removeReceiver`
- `activateFiltering`

The first two methods have to be called before executing. Otherwise an `IllegalStateException` will be thrown. All these methods cannot be called after executing. A try will result in the same exception.

The `configure` method takes the query information. The language of the key can be omitted.

Each provider demands at least one receiver. Calling `addReceiver`, then `removeReceiver` and executing the provider will throw an `IllegalStateException`. 

Listing 6.2: Example Configuration File. The values of the keys have to be inserted.
A call of `activateFiltering` instructs the provider to register each found image URL and provide the image, only if it was not provided before.

Executing means to call the `execute` method. It will return immediately. The providing runs concurrently in an own thread.

### 6.3 Receiving Images

Listing 6.1 uses the `FileWritingReceiver` to receive all web images. It writes all received images to the system’s temporary directory by default. But another directory can be given in the constructor. This receiver is only an example and should be replaced by a new implementation. It is the place for the computer vision algorithm. If it receives an image via the `processWebImage` method, it gets the web image and its source, the provider, as reference. If the `AllInOneProvider` or the `LanguageIterator` is used, the source will be one of the many providers created by these two special ones. And each of these many sources will send a finish signal.

The received `WebImage` provides the following information:

- the image file as byte array,
- the URL, from which the image was downloaded,
- the search key, with which the image was found, and
- the rank of the image, which is the position in the result list of the search.
7 Evaluation

This chapter is divided into two sections evaluating the work done. In Section 7.1 a test is described, which determines if the developed library performs better than a standard script. Section 7.2 compares the precision of nine services exemplarily.

7.1 Performance Test

In this test scenario two programs downloaded a number of images related to a given search key. Their execution time were measured with the shell command `time`.

This procedure was executed ten times for each number of images: 1, 2, 4, 6, 10, 20, 50, 100, 500, 1000. As search key “dog” was chosen, because enough images were found with this word.

The tested script is a shell script with `wget` and `perl` calls. It is printed in Listing A.1. Listing A.2 shows the Java test candidate. Both use Google Images as search engine and have the same timeouts for HTTP connections, without retrying on failures.

Figure 7.1 on page 25 shows the average execution times, which are also listed in Table A.1. The used script performs better, if only one image is to be downloaded. At this stage, the Java library is not able to benefit from concurrency. But with two images it can download concurrently already and terminates earlier. Henceforward, the execution time of the script increases linearly, while the Java program only waits for the latest concurrent download.

This test demonstrates, that the developed library performs better in almost all cases. The overhead generated by Java objects and thread pools is marginal.

7.2 Exemplary Precision Comparison

This section compares the precision of the first twenty images of nine different services. The precision of found images is an important factor for computer vision algorithms. It is defined as the fraction of relevant images in the set of found images. More precisely: If $F$ is the set of found images and $R$ the set of all relevant images, then is:

\[ \text{precision} = \frac{|R \cap F|}{|F|} \]

But, if only found images are observed, then this can be simplified. If $f$ is the number of found images and $r$ the number of relevant found images, then is:

\[ \text{precision} = \frac{r}{f} \]
The question, which images are relevant, is more difficult. Here, a simplified classification is used, that bases on the annotation model of [21]. It describes the three classes *in-class-good*, *in-class-okay* and *non-class* with two sub-classes *abstract* and *non-abstract*. Since only *in-class* and *non-class* matter for the above defined precision, images will be labeled only with these two. This means, that also abstract images like drawings can be in-class.

The compared services are: Altavista, Ask, Bing, Flickr, Google Images, Picasa, Picsearch, Webshots, and Yahoo. Since the intention of this section is to exemplify the precision of services, only the following three animal names were chosen as search keys: dog, giraffe and mouse. Considering the first twenty images mostly represents the first impression of a service user. With these twenty images per search key, sixty images per service are examined.

Table A.2 lists the measured precision for each service and search key. This data is illustrated in Figure 7.2. The results show, that the ambiguous term *mouse* is an important factor. A mouse can be the animal or a computer mouse. Many search results presented the last meaning. The high average precision of Ask is derived from only a few computer mice in the search result. Ambiguous search keys can be defined more precisely. Picasa’s precision for mouse was measured as 10%, but for *mouse+animal* are gained 95%.

If the key mouse is omitted, the average precision for dog and giraffe is in the range from 87% (Flickr) to 100% (Google Images). This demonstrates, that all these services are generally applicable for computer vision algorithms. To get significant data the measurement has to be repeated with more search keys.
Figure 7.1: The results of the performance test.

Figure 7.2: The measured precision values besides the average precision.
Figure 7.3: The first twenty dog images returned by Google (random order).

Figure 7.4: The first twenty giraffe images returned by Google (random order).
8 Summary

This thesis gives an overview about the retrieval of image data via image search engines and online photo albums. A Java library has been developed, which provides a common interface to access three different services.

Each service can be called with translated search keys automatically. This method is able to download far in excess of 100,000 images per search key. Besides, it is designed to be easily extendable. New services can be added very fast.

The following techniques of many retrieval systems are merged in the library:

- The position of an image in the search results can be a measurement of correlation to the given search key. It is provided by the common interface.
- Translating the search key to other languages can multiply the number of found images. 51 languages are supported.
- Different portals in different countries of one service can lead to different results. This is utilized by one image provider. The other services do not return different results.

Chapter 7 demonstrates that the developed library performs better than a standard script and retrieves thousands of images in a few minutes. In the second part is seen that the precision of the evaluated services is sufficient for computer vision research. The considered services are suitable to implement via the web image retrieval library.

With the solutions given in Chapter 5 the most web image retrieval scenarios are supported. The library generalizes the problem of image retrieval expediently. In the future, the development of a well-engineered data type for web images could enhance the project. But the library is limited by its content-independent approach. A second library merging content-dependent knowledge, for example common filters, could use this work and simplify the development of new computer vision algorithms.
Bibliography


A Additional Material

A.1 Performance Test

Listing A.1: Short shell script.

```java
public static void main(final String[] args) {
    if (args.length != 2) {
        System.exit(1);
    }

    final String searchKey = args[0];
    final int numberOfImages = Integer.parseInt(args[1]);
    final WebImageReceiver receiver = new FileWritingReceiver(".");
    final WebImageProvider provider = new GoogleImageProvider();
    provider.addReceiver(receiver);
    provider.configure(searchKey, numberOfImages);
    provider.execute();
    provider.waitForEndOfProviding();
}
```

Listing A.2: The test candidate using the Web Image Retrieval API.
Table A.1: Average execution time of ten iterations.

### A.2 Precision Statistics

<table>
<thead>
<tr>
<th>service</th>
<th>average</th>
<th>dog</th>
<th>giraffe</th>
<th>mouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altavista</td>
<td>0.73</td>
<td>0.95</td>
<td>0.90</td>
<td>0.35</td>
</tr>
<tr>
<td>Ask</td>
<td>0.98</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Bing</td>
<td>0.68</td>
<td>0.90</td>
<td>0.95</td>
<td>0.20</td>
</tr>
<tr>
<td>Flickr</td>
<td>0.80</td>
<td>0.95</td>
<td>0.80</td>
<td>0.65</td>
</tr>
<tr>
<td>Google</td>
<td>0.87</td>
<td>1.00</td>
<td>1.00</td>
<td>0.60</td>
</tr>
<tr>
<td>Picasa</td>
<td>0.67</td>
<td>1.00</td>
<td>0.90</td>
<td>0.10</td>
</tr>
<tr>
<td>Picsearch</td>
<td>0.75</td>
<td>0.90</td>
<td>0.95</td>
<td>0.40</td>
</tr>
<tr>
<td>Webshots</td>
<td>0.70</td>
<td>1.00</td>
<td>0.85</td>
<td>0.20</td>
</tr>
<tr>
<td>Yahoo</td>
<td>0.82</td>
<td>1.00</td>
<td>0.90</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Table A.2: The measured precision of the first twenty images.
Figure A.1: The measured precision values of each service.

Figure A.2: The measured average precision of each service.
Figure A.3: The measured average precision of the search keys dog and giraffe.
B Implementation Notes

B.1 The Web Image Retrieval API

The developed library is written Java 6. For the Flickr service the flickrj API, developed by Anthony Eden, Martin Goebel, Matthew Ray, Matthew MacKenzie and Till Krech is used. Writing the Google implementation was inspired by Java scripts from Marco Kortkamp and C# scripts from Ilan Assayag. \(^1\)

B.2 Evaluation Scripts

For evaluation purpose, a shell script with wget and perl calls was written. The repeated calls with the `time` command and building average values carried another shell script. Gnuplot generated the plots from these data sets.

B.3 Used Software

The following software has been used to write source code, generate files and convert data.

- Sun Java JDK 6
- Netbeans 6.7
- Checkstyle plugin by Petr Hejl
- UMLet
- \LaTeX
- BibTeX
- Texmaker
- Kile
- JabRef
- Bash

\(^1\)http://www.codeproject.com/KB/IP/google_image_search_api.aspx
• Bc
• Wget
• Perl
• Gnuplot
• ImageMagick

Thanks to all the people who supported the development of the mentioned software.
C Declaration of Independent Work

I declare that this thesis is my own work and that I did not use any other resources than the denoted ones. Also the contained data sets, figures and program code were composed by me or taken from the denoted sources.

Bielefeld, November 27, 2009

Maikel Linke