Semi-Supervised Image Classification based on a Multi-Feature Image Query Language

Raoul Pascal Pein

A thesis submitted to the University of Huddersfield
in partial fulfilment of the requirements for
the degree of Doctor of Philosophy

The University of Huddersfield

September 2010

I dedicate this thesis to Prof. Frank Heubach who made all this possible.

I would like to thank everybody who gave me support in finishing this thesis. Special thanks go to my supervisors Dr. Joan Lu and Prof. Dr. Wolfgang Renz for their specialist support and to family and friends for their moral support.

Contents

List of Tables

List of Figures

Abstract

The area of Content-Based Image Retrieval (CBIR) deals with a wide range of research disciplines. Being closely related to text retrieval and pattern recognition, the probably most serious issue to be solved is the so-called "semantic gap". Except for very restricted use-cases, machines are not able to recognize the semantic content of digital images as well as humans.

This thesis identifies the requirements for a crucial part of CBIR user interfaces, a multimedia-enabled query language. Such a language must be able to capture the user's intentions and translate them into a machine-understandable format. An approach to tackle this translation problem is to express high-level semantics by merging low-level image features. Two related methods are improved for either fast (retrieval) or accurate (categorization) merging.

A query language has previously been developed by the author of this thesis. It allows the formation of nested Boolean queries. Each query term may be text- or content-based and the system merges them into a single result set. The language is extensible by arbitrary new feature vector plug-ins and thus use-case independent.

This query language should be capable of mapping semantics to features by applying machine learning techniques; this capability is explored. A supervised learning algorithm based on decision trees is used to build category descriptors from a training set. Each resulting "query descriptor" is a feature-based description of a concept which is comprehensible and modifiable. These descriptors could be used as a normal query and return a result set with a high CBIR based precision/recall of the desired category. Additionally, a method for normalizing the similarity profiles of feature vectors has been developed which is essential to perform categorization tasks.

To prove the capabilities of such queries, the outcome of a semi-supervised training session with "leave-one-object-out" cross validation is compared to a reference system. Recent work indicates that the discriminative power of the query-based descriptors is similar and is likely to be improved further by implementing more recent feature vectors.

Chapter 1.

Introduction

With rapid development of digital technologies, building an efficient and reliable image retrieval system is always challenging in computing science and related application disciplines [14, 15, 38, 44, 47, 80, 94, 116, 145]. A typical application area of CBIR is multimedia publishing and design. Often, an image with specific properties is required for a certain layout. Garber and Grunes [44] describe a typical layout task, where somebody needs to pick some images from a huge repository. Similarly, the growing amount of personal digital image collections (like Flickr [145] or Picasa [47]) could benefit from a CBIR system. Another suggestion is the use of CBIR techniques to retrieve copyright infringements [116]. Further more, in medical environments, the use of 3D body scanners with high resolution results in an immense amount of visual data [80]. Space agencies and Geographic Information System (GIS) companies taking pictures with high-resolution cameras on satellites also produce large amounts of pixel images, depicting planet surfaces, surface features and other content [14, 38]. Even more general cases in Multimedia Information Retrieval (MIR) have been studied, such as the powerful supporting tools in the retrieval of 3D models for engineering companies [94] or similar sound files for musicians [15]. It follows that this key technology, Content-Based Image Retrieval, always plays a significant role in the application search engines, though the development of consistent theory in CBIR is still rudimental.

Many basic issues in CBIR have been collected by Eakins and Graham [34] in 1999. The most severe problems of image retrieval identified in their report remain unsolved. The key is "bridging the semantic gap" between low-level image content (pixels, as seen by machines) and its high-level meaning (semantics, as seen by humans) [149]. Though many new technologies and methods have been continually improving the quality of Multimedia Information Retrieval (MIR) [140] and several real-life applications have been developed in many areas. A summary of recent challenges in MIR is provided by

Lew et al. [70].

MIR systems mostly apply the feature vector paradigm because the documents to be retrieved are typically very large. Further, the information contained in each document exhibits a considerable amount of redundancy and fuzziness. For example, several default colour space models in computing are mapped to 24 bits which corresponds to $2^{24} = 16777216$ different colours for each pixel. Modern hardware is capable of generating images with a resolution expressed in megapixels. Therefore, a single image can easily contain millions of pixels, encoding some real or synthetically generated information. Matching any set of two high-resolution images directly would consume a very high amount computing resources, rendering a naive matching based retrieval for thousands or even millions of images useless with the currently available technology. To greatly reduce the retrieval complexity, MIR systems perform the most time consuming part of data analysis only once and generate a so-called "feature vector" for each document during the indexing phase. These feature vectors are designed to contain a highly condensed piece of information representing the original data, without losing too much of the relevant characteristics. A direct comparison between two feature vectors is expected to be similar as a comparison of the respective original documents. The feature vector approach is basically a lossy transformation from an extremely high dimensional representation to a representation with a dimensionality that can be handled within seconds.

Gupta and Jain [50] coined the term Visual Information Retrieval (VIR) which covers the visual subset of MIR. They propose a retrieval system to find images and videos. VIR is a research domain that links the analysis aspects of computer vision with the querying aspects of database systems. A main difficulty in their work was to map natural language to a machine understandable query language. Natural language may be suitable to describe complex sceneries, but it is inherently ambiguous and the retrieval has to be performed by a machine with no deeper understanding. Also a plain textual query is considered to be not powerful enough to express all required details. In contrast to VIR, the research for other media types such as audio data are less common. One such system specifically developed to retrieve music is "Muugle" [15].

After more than a decade in research on CBIR and MIR in general, the interest in this area is very high and still seems to increase [28]. Observing the currently available software compared to the amount of publications indicates that still no general breakthrough has been achieved. A study by Datta et al. [28] published in 2005 analyses the amount of related publications in detail. Only about 20% of about 300 briefly surveyed papers in this study described real-life applications, most of them being merely proto-

types. This seems to be an indication that there are many theories available, but none is capable of causing a real breakthrough. Thus, Datta et al. [28] recommend the building of useful systems in parallel, even if they are limited to specific domains.

1.1. Research Hypothesis

This thesis investigates several CBIR techniques and their interrelationship. The main emphasis is put onto the issue of querying. Being able to express the user's needs in a machine-readable way is assumed to be a crucial element of any retrieval system. Generic object recognition and semantic scene interpretation remains an open problem [70]. Thus, the so-called "semantic gap" cannot be completely bridged yet. This leads to the research question investigated in the following:

> Are multi-feature query languages capable of narrowing down the semantic gap in Content-Based Image Retrieval?

Below, a multi-feature query language is evaluated to estimate its capabilities in mapping high-level semantics to a set of low-level feature vectors. A machine learning approach is applied to generate a set of the most efficient queries to find specific categories within an image repository. The discriminative power of each query is evaluated.

1.2. Contribution

- Two feature merging approaches for fast retrieval and for image categorization/ tagging (sections 3.6.2 and 3.7.9)
- A way of describing high-level semantics with an arbitrary and extensible set of low-level features in a query language (sections 3.7.9 and 6.5)
- A machine learning approach to find queries describing a category with positive and negative Query-By-Example (QBE), thus potentially reducing the semantic gap (sections 3.7.9 and 6.5)
- The derived queries describing a category are comprehensible to humans and thus can be modified manually (sections 3.7.9 and 6.4)

- Proposed metrics for evaluating the appropriateness of feature vectors and their potential information gain in various applications, e.g. which available feature vector achieves the best ranking quality for a given image category (section 6.3)
- Evidence that the expressiveness of the query language developed by the author of this thesis can compete with traditional machine learning approaches (section 6.6)

1.3. Roadmap

The thesis is arranged in seven chapters. The background research in chapter 2 is split into four smaller sections. Publications in multiple CBIR related research areas are mentioned in section 2.1. Related preliminary work of the author of this thesis is described in section 2.2. Section 2.3 provides an overview of currently unsolved problems to motivate the aims and objectives of this thesis (section 2.4).

Chapter 3 lists the methodologies used in this thesis inferred from the background research. The design and implementation of the prototype developed to support this thesis are explained in chapters 4 and 5.

The core of the thesis is located in chapter 6. This chapter focuses on six different case studies that are used to underpin the methodology from chapter 3. The first case studies examine several basic feature vector properties an their interrelationship (sections 6.1 to 6.3). The latter ones focus on the machine learning aspects and how low-level feature vectors could be applied to describe higher-level semantics (sections 6.4 to 6.6).

The final chapter 7 concludes the thesis and gives a brief overview of the achievements of this thesis and potential future work in this particular area.

Chapter 2.

Background Research

2.1. Related Work

Many of the basic issues in CBIR have been collected by Eakins and Graham [34] in 1999. The most severe problems of image retrieval identified in their report remain unsolved. The key to image retrieval is "bridging the semantic gap" between low-level image content (pixels, as seen by machines) and its high-level meaning (semantics, as seen by humans) [149]. In Figure 2.1, the semantic gap is represented by the two research fields *Features/Similarity* and *Annotation*. The first one deals with low-level content whereas the second one deals with its high-level counterpart. A technology to connect these two fields is the *Categorization*.

New technologies and methods have been continually improving the quality of MIR [140] and several real-life applications have been developed in many areas. Yet, a large amount of recent challenges in MIR are still to be solved. Lew et al. [70]. provide a summary of these challenges.

Currently the research seems to shift from image retrieval towards video retrieval. Nevertheless it should be considered that the diversity in multimedia retrieval is beneficial for all sub-disciplines as they overlap in many cases. Also the researchers coming from several different disciplines contribute to the richness of different approaches and solutions [139].

The recent overview provided by Datta et al. [30] concludes with the statement that the research focused more in *systems*, *feature extraction* and *relevance feedback* than in application-oriented aspects such as *interface*, *visualisation*, *scalability* and *evaluation*. Thus it seems desirable to improve research in these areas.

According to Lew et al. [70] there are several recent research topics trying to bridge the semantic gap. In human-centred computing the system tries to satisfy the user while

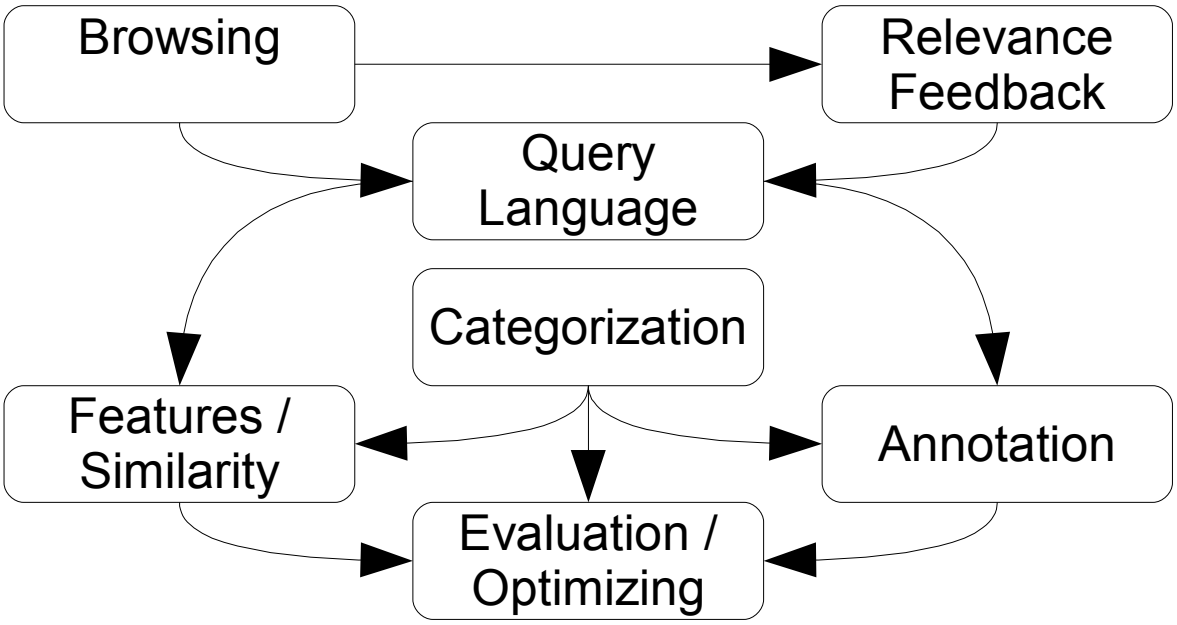


Figure 2.1.: Research Field Dependencies

keeping the interface easily understandable. Learning algorithms could be beneficial in adding semantic value. Developing new features based on low-level information may still be beneficial, when adapted to the human perception and easy to use. Their conclusion is that none of the major challenges in MIR are actually solved and all areas require significant further research.

This section briefly describes several research areas closely related to CBIR (fig. 2.1). Some of these are user-centred and some deal with the underlying methodology. Concerning the user interface, the areas of *Browsing*, *Query Languages* and *Relevance Feedback* play an important role. This supports the user in creating queries and to navigate through a given repository. Creating *Feature Vectors* and *Similarity Measures* is a matter of improving the system quality on the server side. Similarly, the *Annotation* is important to enable keyword based retrieval. They usually need to be tuned by an *Evaluation* process. The area of *Categorization* is an approach to link existing low-level features directly to high-level keywords and categories. Finally, there are *Retrieval Framework Designs* available, which provide researchers with the basic functionality needed to do this kind of research.

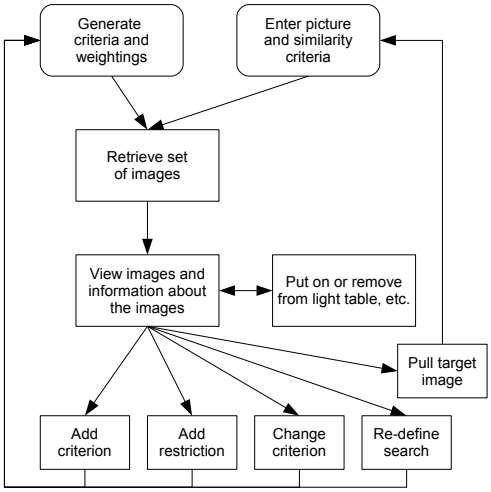


Figure 2.2.: Searching Process [44]

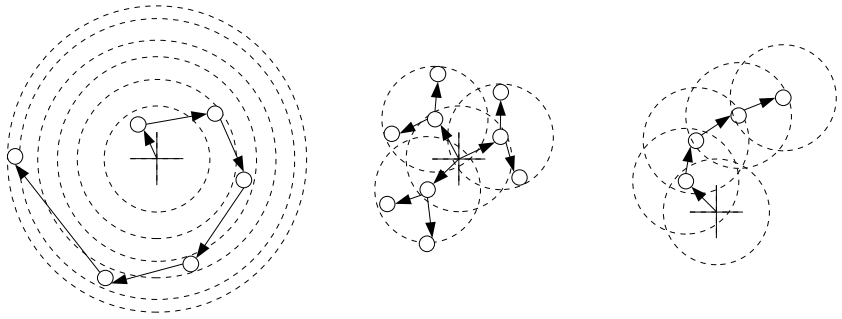
2.1.1. Browsing

In order to build a user interface for a CBIR system, typical related work flows need to be examined. A detailed analysis of a retrieval work flow is described by Garber and Grunes [44]. They focused on a common task of art directors, searching images for use in advertisement. The iterative searching process is depicted in figure 2.2.

A perfect search engine would already contain all relevant hits in this first overview, omitting all unrelated content. In reality, *precision* and *recall* are far from being perfect and the user needs to pick the relevant images manually. At this point, a well designed browsing interface could guide a user through the repository, presenting only the most related fraction of images and refining the initial query. This iterative approach supports an evolving search process as described by Garber and Grunes [44].

Frohlich et al. [43] interviewed 11 families about their photo management and usage habits. Their paper concludes with a list of requirements for useful software. A similar use case was examined by Rodden and Wood [107]. Their analysis of user behaviour was carried out for six months and 13 participants using the management system "Shoebox". The outcome was that the advanced features such as CBIR were rarely used. Instead, the basic browsing capabilities proved to be the most important feature of the system. This was mainly due to the rather small and well known personal image set.

The recent system PARAgrab by Joshi et al. [63] provides a browsing interface that offers several retrieval techniques. Results are listed by matches on file name, surrounding text and images that have been viewed by other users. In addition, the semantics



(a) Strict order (b) Hierarchical tree (c) Search sequence

Figure 2.3.: Search Strategies [63]

from WordNet [76] are used for improving the keyword based search. A ranking based on visual similarity and query-by-example rounds off the system.

Shneiderman et al. [122] focus on a user interface, determined to provide simple browsing and using as much meta information as possible. Sharing the photos with others is considered important, but it is assumed, that the casual user is less interested in annotating images than browsing them to have fun. Otherwise solutions like the direct annotation in PhotoFinder [121] could utilise their full potential. Huynh et al. [55] propose a way to organize images by their timestamps and a ranking simultaneously. The time line is broken down into parallel parts and the images are located in the correct position. To stress the relevance of each hit, the images are scaled accordingly. Several techniques to interweave multimedia results from several dimensions are discussed by Candan et al. [18].

2.1.1.1. Browsing Strategies

Some basic browsing strategies are described by Joshi et al. [63]. The concepts are visualised in fig. 2.3. All these strategies should be supported by a retrieval client.

(a) "Begin with a single query image and browse result pages strictly in the order of visual similarity to query."

(b) "Begin with a single query image and then hierarchically explore the next level, using each top level result as a subsequent query."

(c) "Begin with a single query, and keep performing visual searches based on personal preference and relevance to query. The sequence thus can potentially deviate further and further from the original query."

Especially in a creative process, the user is usually not sure about the desired findings and the requirements may change during a retrieval session. This probably matches best browsing strategies b and c. During this process, the user might want to keep some intermediate results as a potential candidate and put them onto a "light table". Later, these images can either be used directly or fed into a new search, like in strategy b.

2.1.1.2. Visual Arrangement of Search Results

Table 2.1.: Thumbnail Layouts

Article	**Dim.**	**Pattern**
traditional	1	list
Liu et al. [71]	2	Similarity, Adjustable Overlap
Liu et al. [71]	2	Fisheye
Huynh et al. [55]	2	Truncated Timeline, Relevance Zooming
Torres et al. [133]	2	Concentric Rings
Torres et al. [133]	2	Spiral
Torres et al. [133]	2	Spiral + Image Similarity Degree

Liu et al. [71] examined several layouts for arranging result thumbnail images on a screen. The images were sorted by similarity, either in a 1D ranking list or clustered in a 2D grid. They performed a user study to evaluate the influence of image overlapping, zoomed images and different arrangements. Huynh et al. [55] and Torres et al. [133] also experimented with innovative 2D layouts. Some proposed layouts are listed in table 2.1.

First of all, it is to be decided, in how many dimensions the results are to be displayed. The traditional solution is a simple one dimensional ranking, beginning with the highest and proceeding with a gradually decreasing similarity. This approach requires an algorithm to determine a linear ranking.

Exploiting two dimensions for display, greatly increases the possible layouts. The resulting images could be directly arranged by 2 features on an x and y axis, which hopefully generates meaningful clusters. Having a single ranking dimension, the images could be arranged in a zigzag pattern, a ring or spiral with the best match in the centre [133] or even randomly distributed on the screen. Most important hits can also be accentuated by an enlarged image [55, 71]. Clustering the results by semantics provides another guideline for a good layout.

Experiments indicated that an irregular (or "untidy") arrangement of overlapping thumbnail images is rather irritating than helpful. Sorting the images into a clean grid was much more accepted in multiple studies [71, 108].

Having arranged several thumbnail images on the screen, the user might want to see some meta information. This could be done by a dedicated area at the border, a tool tip window or separate windows. A dynamical approach could also be used to reduce the overall amount of images displayed. Clusters of very similar images could be merged into a single representative image, hiding the others. Selecting that image would then expand the view to show all of the hidden images.

Another interesting feature is to crop thumbnails automatically to enhance its information density ans reduce the required space. The proposed techniques by Suh et al. [128] include cropping by salient regions and faces.

2.1.1.3. Thumbnail Format

With unlimited resources, a system could easily store the original images in full resolution and without any compression. This would offer a very high flexibility and the system could scale the images down to an appropriate size for each reading access. In reality, the restrictions of the technology used need to be considered.

Because each stage in the system has several restrictions, a hybrid approach may be beneficial. The browser should have access to Joint Photographic Experts Group (JPEG) or Portable Network Graphics (PNG) images of various resolutions. JPEG is usually preferable for photographs and PNG is more suitable for images with high contrast, like sketches or drawings.

At least two strategies for persistence issues are possible. The persistence could be either optimized on speed or a reasonable storage capacity.

Having almost unlimited (or just very cheap) disk space available, the images could be easily stored in multiple resolutions redundantly. The Database Management System (DBMS) or file system would simply load the relevant images directly from the hard disk and could stream the file directly to the client. The bottleneck would still be the connection between client and server. Thus, the time required for reading the information from the hard disk is unlikely to be perceivable by the user. A disadvantage would be that all the thumbnails need to be prepared in advance.

A more sensible and flexible solution is to avoid redundancy and save disk space by using multi-resolution images. Image formats like Joint Photographic Experts Group 2000 (JPEG 2000), Enhanced Compression Wavelet (ECW), Multiresolution Seamless Image Database (MrSID) or Progressive Graphics File (PGF) formats are all capable of doing such tasks. Only one image needs to be stored. At runtime, the system simply reads data up to the desired accuracy and then stops reading the stream. The main

Table 2.2.: Thumbnail Image Formats - Display

Format	colour depth ($\geq 32bit$)	open standard	browser-support
JPEG [137]	✓	most	✓
JPEG 2000 [130]	✓	most	✗
PNG [31]	✓	✓	✓
GIF [27]	✗	patented by Unisys	✓
ECW [35]	✓	✓	✗
MrSID [72]	✓	patented by LizardTech	plug-in
PGF [127]	✓	✓/LGPL	✗

task is to convert the image into a browser supported format. This hardly avoidable computation time is the main disadvantage of this second approach.

Further, if the images are stored in a reasonable quality, the feature extraction process does not need to access the original images in all cases. Especially global features should still be present in "high-resolution" thumbnail images. Often the original images are scaled down in the pre-processing anyway, like the wavelet approach by Jacobs et al. [58]. The requirements for image display *a)* and storage *b)* are different. For display, well-known standards extend portability while the storage format should focus on storage efficiency.

a) The requirements for the browser representation are focused on user friendliness. All used image formats should be easily displayable in common browsers and tools. The file size should be small while ensuring a reasonable quality. Finally, the image resolution depends on screen size (from mobile phone up to a power wall). A set of several formats is listed in table 2.2.

Regarding the browser support, either JPEG (photographs) or PNG (drawings) should be used. GIF has no real advantage to PNG. All the other formats are simply not supported by most browsers. Nevertheless, they could be of much use in specialized clients.

b) The second task is to store the thumbnail images in a reasonable quality and as many resolutions as possible. There are both lossy and lossless formats available. An interesting feature to be used is *multi-resolution*, allowing the extraction of smaller

Table 2.3.: Thumbnail Image Formats - Storage

Format	comp. algo.	lossy/ lossless	multi-resolution	open standard	focus of optimizing
JPEG [137]	DCT	lossy	✗	most	size
JPEG 2000 [130]	DWT	both	✓	most	quality
PNG [31]	LZ77	both	✗	✓	quality
ECW [35]	DWT	lossy	✓	✓	memory
MrSID [72]	DWT	both	✓	LizardTech	flexibility
PGF [127]	DWT	both	✓	✓(LGPL)	speed

versions of an image without loading the whole file. Some recent formats are compared in table 2.3. Unfortunately, none of the multi-resolution formats are supported by browsers. Allowing a conversion from a storage format into a browser-standard, either ECW or PGF should be used. Being open, both standards can be used. A slight advantage of PGF is the lossless option.

2.1.2. Query Language

Users need to formulate their queries, i.e. be able to express what they expect to receive from a search engine. Converting the mental model to a structured machine understandable format [50] is not straightforward. Firstly, it requires a user, whose mental model correlates to the machine model. Secondly, the query language must be able to capture all the relevant nuances of the users intent. Otherwise the retrieval system cannot return relevant results without making assumptions.

There are multiple languages available which are basically suitable to be applied in CBIR. In the following, a couple of textual and visual query languages are presented.

2.1.2.1. Textual

Most query languages for search engines are based on strings. This is not surprising, as most documents can be retrieved by keywords. It is especially true for pure text documents, but also applies to a certain extent to correctly annotated multimedia data.

Usually these query languages are defined in Backus-Naur Form (BNF) to ensure a mathematical sound background and consistency. Representative examples are widely used in Internet search engines, such as the *Google Query Syntax* [46]. Others are provided by frameworks and toolkits like the *Apache Lucene Query Syntax* [7]. Often these languages understand basic queries like single or multiple keywords separated by

Table 2.4.: Language Approaches

Language	CBIR Approach	Base Language
Nepal and Ramakrishna [84] FOQL	object driven	ODMG/OQL
Town and Sinclair [134] OQUEL	ontology driven	none/natural
Pein et al. [95]	feature driven	Lucene

spaces which are widely used by untrained users. More advanced searchers are also able to exploit meta information (e.g. *title*, *author*, *creation date*, etc.), Boolean expressions and nesting. While there is a diversity of full text query languages, most of them follow this basic behaviour and even define a similar syntax. Nevertheless there seems to be no uniquely accepted standard.

Still there are several standards of query languages available. This is especially true for areas, where a high accuracy is of importance. A popular language used in relational data bases, is the SEQUEL/SQL family [21]. It can be applied for huge amounts of structured data and ensures a short response time combined with absolute and deterministic precision. Recently a new kind of data base is gaining popularity, the XML database. Its main advantages are the increase of XML based communication protocols and the high flexibility to storing new or unpredicted data structures. Much effort is currently put into research for developing a query language for XML databases. A promising candidate is XQuery [20], proposed by the World Wide Web Consortium (W3C). It may become the successor of SQL. It is also attempted to add full text search capabilities to this language, e.g. the extension TeXQuery [5].

In the area of CBIR it is very difficult to point out query languages or even standards. Most languages are not more than a research project. Three representative languages with varying approaches are compared in table 2.4. Example queries for the languages are given below.

FOQL example The Fuzzy Object Query Language (FOQL) by Nepal et al. [83] is specifically designed for image databases. As it is based on Object Query Language (OQL), the syntax complexity is similar to that of SQL. This language lets the user specify anything in detail. An example of a natural language query converted into FOQL is given below (see Nepal et al. [83]):

> "Find all distinct images from the image collection 'Flower' that have color similar to the example image 'flower1.gif' and contain an image component

similar to the example image component rose (whose imgobjno is '01'). Assume an overall similarity of greater than 0.8."

```
select distinct [0.8] I.imagename
from Img-col R, I in R.has, O in I.contains,
    Image J, Image-comp K
where R.name = "Flower" and I.colormatch(J)
    and O.similarto(K) and J.imagename = "flower1.gif"
    and K.imgobjno= "01"
```

OQUEL examples The Ontological Query Language (OQUEL) is an image retrieval language proposed by Town and Sinclair [134] with a focus on user friendliness. It is based on an ontology and the user input remains very basic. The actual work has to be done by the system. The system is working with segmentation and categorization. Regions are arranged in a region graph for spatial relationships. Below some query examples by Town and Sinclair [134]:

- "some sky which is close to trees in upper corner, size at least 20%"
- "[indoors] or [outdoors] & [people]"
- "[some green or vividly coloured vegetation in the centre] which is of similar size as [clouds or blue sky at the top]"
- "artificial objects, smooth and polygonal"

Own language examples The third language by Pein et al. [95] (see section 2.2) is designed to be user-friendly and flexible. The language follows the principles of the Lucene Query Syntax [7] where each term is composed of a an optional field and the desired content. The field is interpreted as a feature and the content is directly forwarded to the corresponding feature parser.

"Find an orange image or a fruit with double weight on the mean colours."

```
fv_mean:($orange$)^2 keywords:"fruit"
```

"Find images that are wavelet-similar to image 123 and remove images that have more than a 0.9 histogram similarity to image 987."

```
fv_wavelet:123 NOT fv_histogram:987@0.9
```

Table 2.5.: Language Features Compared

Language	Fuzzy Boolean	Min Threshold	User Defined Sorting	Extensible (Features)	AND-OR-NOT	Weights	High-Level Concepts	Simple Structure	Query-By-Example
FOQL [84]	✓	✓	✓	✓	✓	✓[1]	✓[2]	✗	✓
OQUEL [134]	✓	✓	✗	✗[3]	✓	✓	✓	✓	✗
Pein et al. [95]	✓	✓	✗	✓	✓	✓	✗[4]	✓	✓

[1] sum of partial weights must be 1.0
[2] keyword *define* to map low-level queries to high-level concepts
[3] ontology can be modified
[4] mapping only possible by nesting/meta features containing prepared low-level queries

Comparison Table 2.5 compares the three query languages mentioned above concerning several key CBIR properties.

The *Fuzzy Boolean* is necessary to capture the blurred line between hit and miss. To limit the result space to a reasonable amount, the *Min Threshold* is required. Both are available in all checked languages. A *User Defined Sorting* is helpful to further control the ranking, especially in situations with many identically ranked images, but this is only supplied by FOQL. For a widespread use in several scenarios, the *Extensibility by Features* seems to play an important role. While OQUEL requires the modification of the ontology model, the others capture new features individually. Boolean combinations by *AND-OR-NOT* and *Weights* are supported in all cases.

The user friendliness is assessed by three final properties. Introducing important *High-Level Concepts* into the language by Pein et al. [95] requires additional work in the feature code itself, while FOQL offers the keyword *define* and OQUEL implicitly uses high-level concepts. Measuring the *Simple Structure* is indeed arguable. In this case, it is checked, whether the language is initially designed for amateur users or for specialists. FOQL requires much overhead, the other languages offer very short and concise queries. Finally the integration of Query-By-Example (by ID or Uniform Resource Locator (URL)) is not available in OQUEL.

2.1.2.2. Visual

Providing a graphical query interface to the user can simplify the assembly of queries. Rather than having to type a query string, the query can be assembled with a couple of mouse-clicks. These interfaces are similar to a drawing program for diagrams. The query is drawn to a canvas and all available constructs are listed in a tool bar. The user picks the relevant nodes and drops them at relevant positions. Each node could be edited in detail or linked to other nodes while each single element has certain semantics. In the end the user has built a graph representing the query, which is sent to the search engine.

Several visual query languages have been developed for GIS environments [14, 17, 38, 88]. Others are usually related to object oriented approaches [19]. The Classification Query Language (CQL) by Järvelin et al. [60] is based on classifications and their relationship. It provides a user interface with several forms to compose a query.

The proposal by Keim and Lum [64] is a visual interface for a Multimedia Database Management System based on relational data. It differs from visual interfaces in common DBMS, because multimedia content is inherently ambiguous. Its purpose is to simplify the SQL like language towards usability of natural language. Users can see all possibilities and choose them by point-and-click (no misspelling).

2.1.3. Relevance Feedback in Search Results

Relevance feedback in CBIR applications gives the user some additional control to influence the search results. Usually the user can judge the quality of a result set and submit hints to the system in order to improve future retrieval sessions. A relevance feedback mechanism needs to be integrated into the result browser. It relies on the active participation of a searcher.

Zhou and Huang [151] published a useful overview of multiple relevance feedback aspects. The algorithms are classified into several categories for short-term and long-term learning. They summarized several issues to be considered when designing a relevance feedback algorithm as follows (see Zhou and Huang [151]):

- 'Minimize the influence of (false) negative examples'
- 'The training set must be large enough to cover all feature dimensions'
- 'Pre-clustering usually requires to assume a specific point-of-view'

- 'Queries can be considered to be global or regional'
- 'The low-level features should be enriched with textual annotation'
- 'The nearest neighbour search needs to be fast, even with many dimensions involved'

Many papers related to relevance feedback in CBIR scenarios can be found. Some interesting contributions are recommended for further reading. An early article by Benitez et al. [12] gives a good idea of the basic problems involved. Müller et al. [77] discuss the differences between positive and negative relevance feedback.

A couple of research systems using relevance feedback have been developed. One of the first systems was ImageRover by Taycher et al. [131], which is an early image retrieval system with integrated relevance feedback for Internet use. Sciascio et al. [118] additionally allowed query-by-sketch in their approach. A more recent approach is Cortina by Quack et al. [100], a large-scale CBIR system which also included relevance feedback aspects. One of the latest researches in this area has been published by Chiang et al. [24]. They combined relevance feedback with an object movie retrieval.

A special variation of relevance feedback is the annotation by searchers. Instead of letting the originator of the image do all the hard work, it is delegated to all users. Russell et al. [110] developed the LabelMe system, which is suitable for region aware web based image annotation.

2.1.4. Features and Similarity Measures

The selection of supported features and similarity measures is highly application specific [28]. For this reason it should be possible to add them as needed to the retrieval engine used [90].

Many image features have been proposed so far. While earlier ones focused on single low-level features, later approaches tried to include more and more complex ones. Also the effort shifted from global to local features. In the following some exemplary features are described.

Jacobs et al. [58] proposed a feature based on Haar wavelets. Their approach is performing a multi resolution decomposition of the images. Short signatures containing a set of wavelet coefficients with the highest amplitudes for each picture are extracted. Several other attempts are also based on wavelets (e.g. [32]). Their strength is usually their sensitivity for the image texture.

Histogram approaches are a way to focus on colour-based similarity as well as providing rotation invariance. An examples is the feature vector proposed by Al-Omari and Al-Jarrah [4], compressing the histogram into 12 stochastic moments (mean, variance, skewness, colour correlation) for three colour channels. Another one has been proposed by Berens et al. [13]. Their feature vector compresses an opponent colour histogram with several transformations (Karhunen–Loêve, discrete cosine, Hadamard and hybrid) with a stated compression rate of up to 250:1.

A third field tries to get the grips on shape detection often based on an autocorrelograms. Typical instances are both the features by Latecki and Lakämper [66] and Mahmoudi et al. [73]. Other research even attempts to do some object recognition based on extracted shapes [11] by matching the contours of objects identified in an earlier processing stage. A comprehensive overview of early descriptors for the MPEG-7 standard are presented by Manjunath et al. [74].

One representative for a more recent feature is the aesthetics measure by Datta et al. [29]. They do not try to find images based on similarities, but on higher visual aspects. This feature could be used as a filter to automatically reject poor quality images in advance.

Carefully choosing appropriate features for a use case is only the first step in designing a CBIR application. Afterwards it is to be considered, how the similarity between each of two image features can be calculated. Several basic metrics and their properties are collected and described by Santini and Jain [111] as well as Jolion [62]. It is argued that the similarity (or dissimilarity) should not always be measured as a simple geometric distance. Santini points out that the four axioms for metrics (*self-similarity*, *minimality*, *symmetry* and *triangular inequality*) cannot be applied to both *perceived* and *judged similarity*. They state that several recognition experiments revealed the weaknesses of the distance approach. Still measures like varieties of the Minkowski/L_m distances are widely interpreted as similarity. This is probably mostly done in order to minimize the effort to develop a whole new indexing structure for faster retrieval.

A major problem of many features is their retrieval performance. Having a simple structure and similarity metric, indexes can be based on generic solutions. Using the euclidean distance, allows for implementing one out of several multi dimensional index trees. The more specific a single feature gets, the more specific is the related index structure.

Early research in this field has been done by Seidl and Kriegel [119]. Recent results by Gelasca et al. [45] already claim to handle image databases with more than 10 million

Table 2.6.: Similarity Measures

Citation	Similarity Measure	Feature	Data Format	Maths Model ($s_{QI} = \ldots$)[1]	Symmetric
Al-Omari and Al-Jarrah [4]	vector dot product	histogram moments	vector (12 dim)	$\frac{2(V_{kf}^{Q} \bullet V_{kf}^{I})}{V_{kf}^{Q} \bullet V_{kf}^{Q} + V_{kf}^{I} \bullet V_{kf}^{I}}$	✓
Belongie et al. [11]	L_1 Distance[2]	polygons	vector (3 dim)	$\sum_{i=1}^{n} \lvert Q_i - I_i \rvert$	✓
Berens et al. [13]	vector dot product	compressed histogram	vector (n)	$\left\lVert \underline{q_Q} - \underline{q_I} \right\rVert^2$	✓
Do and Vetterli [32]	Kullback-Leibler divergence (KLD)	wavelet coefficients	18 values	$\sum_{j=1}^{B} D(p(.;\alpha_Q^{(j)};\beta_Q^{(j)}) \lVert p(.;\alpha_I^{(j)};\beta_I^{(j)})$	✗
Jacobs et al. [58]	average colour sum of matches	wavelet coefficients	coefficient bins	$w_0 \lvert Q[0,0] - I[0,0] \rvert - \sum_{i,j:\widetilde{Q}[i,j]\neq 0} w_{bin(i,j)}(\widetilde{Q}[i,j] = \widetilde{I}[i,j])$	✗
Latecki and Lakämper [66]	integral	polygons	tangent function	$\left(\int_0^1 (T(Q)(s) - T(I)(s) + \Theta_0)^2 ds\right) max(l(Q), l(I)) max\left(\frac{l(Q)}{l(I)}, \frac{l(I)}{l(Q)}\right)$	✓
Mahmoudi et al. [73]	L_1 Distance	edges	matrix (n,m)	$\sum_{i=1}^{n} m \lvert Q_i - I_i \rvert$	✓

[1] The value s_{QI} is the similarity between query Q and image I respectively its features.
[2] This is the second stage in the calculation to combine the more complex results"shape context", "appearance cost" and "bending energy".

entries.

Jain et al. [59] mention the problem of score merging when using multiple sources in biometrics. Each source would generate a different score depending on the technology and matching algorithm involved. Merging the results of multiple image unrelated feature vectors in a CBIR system basically faces the same challenges.

Table 2.6 shows a concise comparison of diverse similarity measures. The value s_{QI} defines the similarity between the query representation Q and any image/document representation I. Below the notation of the related maths models are explained.

The similarity equation by Al-Omari and Al-Jarrah [4] uses the two 12 dimensional vectors called V_{kf}. The "$\bullet$" denotes the inner or dot product.

The feature vector proposed by Belongie et al. [11] is matching sets of polygon points based on three values: "shape context", "appearance cost" and "bending energy". Each one is calculated by a different function, but the resulting 3 dimensional vectors Q and I are then merged into a single value.

In the vector dot product of Berens et al. [13], the value $\underline{q}$ represents the vector of weights generated by the histogram compression algorithm. The similarity between those weights is proven to be equivalent to the similarity of the original histograms.

The KLD based similarity measure of Do and Vetterli [32] sums all distances D from all analyzed sub bands j into a single value. The values $\alpha^{(j)}$ ("scale paramater") and $\beta^{(j)}$ ("shape parameter") are defined as the extracted texture features from the wavelet sub band j. Both are assigned to a feature vector $p()$.

In the feature vector by Jacobs et al. [58], w denotes the relative weight of a single value. The notation $X[i,j]$ describes a single two-dimensional coefficient for image X. Quantized coefficients are written as $\widetilde{X}$.

The equation by Latecki and Lakämper [66] uses the tangent function $T(X)$ of the turning function X. Further, $l(X)$ is the relative arclength of an arc and Θ_0 is a constant minimizing the integral.

Mahmoudi et al. [73] also use the L_1 distance for the extracted matrices. The similarity is measured by the summed distance of all value pairs.

2.1.5. Annotation

The annotation carried out by humans is an important way to tell computers something about the *ground truth* of images. In the end, every learning algorithm requires some input for validation. In the simpler case, a search engine can directly use existing

annotation information for text based retrieval [149]. A current meta data standard is the International Press Telecommunications Council - Information Interchange Model (IIM) (IPTC) description, which will most likely be succeeded by the more flexible Extensible Metadata Platform (XMP) [56].

The annotation itself can be done for whole image files or bounded objects within. Simple annotation for each image is relatively straightforward and can be easily supported. Several commercial or experimental systems adopt this scheme, especially for private photo books [107, 145]. Advanced user interfaces may even provide drag&drop solutions [121, 122].

The support of bounded objects requires much more effort, as the regions need to be defined somehow. These regions are usually represented as rectangles, polygons or pixel-based overlays. Algorithms are able to find several regions in an image. Based on those regions, humans only have to select and maybe edit them to define the proper boundaries. The next logical step is to annotate each image region separately [110].

2.1.6. Evaluation of Retrieval Systems

As soon as a CBIR prototype has been developed, the question arises how to evaluate its quality. There are several synthetic benchmarks available.

2.1.6.1. Benchmarks

A very popular, yet non standardized benchmark is using the *Corel Stock Photo* collection. It contains several images, already assigned to several categories. This data is especially useful to measure classification algorithms.

Other than the Corel dataset, the Caltech 101 collection [40] has been specifically developed for benchmarks. It consists of 101 categories (plus a background category) with about 40 to 800 images per category. In addition, the outlines of each object and further annotation are available. Recently an updated version of the collection has been released, named Caltech 256 [48]. The amount of categories has been increased to 256 and each category now holds at least 80 images. Another Caltech database contains several thousand images of planes, motorbikes cars and general background [41].

Further image databases are the UIUC Image Database for Car Detection [2, 3], the TU Darmstadt Database (formerly the ETHZ Database) [67], the TU Graz-02 Database [86, 87] and the MIT-CSAIL Database of Objects and Scenes[132]. The PASCAL Object Recognition Database Collection uses some of the collections above to compile different

test sets used in the Visual Object Classes Challenges [36, 102]. Some collections also contain images from Google, Flickr and the Microsoft Research Cambridge database.

The Benchathlon framework [79] is another attempt to generate and collect benchmarks for a fair and general comparison of various retrieval systems, such as BIRDS-I [49].

Several conferences (e.g. ImageCLEF [25], TRECVID [52, 123]) are especially dedicated to competitive multimedia retrieval solutions. These conferences develop challenging tasks closely related to real-world applications. Each participant needs to submit a prototype system solving certain tasks and the results are then compared to each other.

ImageCLEF [25] is focused on Cross-Language Information Retrieval (CLIR) and CBIR. In the last workshop, the topics were: "photographic retrieval", "medical retrieval", "photographic concept detection", "medical automatic image annotation" and "image retrieval task from a collection of Wikipedia images". The TRECVID [52, 123] is essentially dedicated to video retrieval.

2.1.6.2. Metrics and Analysis

The performance of CBIR systems is usually measured by calculating values as Precision-Recall (PR), Receiver Operating Characteristic (ROC), F-Measure/F-Score and Fall-Out. They use the amount of true/false positives and true/false negatives in the result set. Some metrics are dependent on other values like the result size or cannot be reduced to a single number. Each metric must be applied carefully regarding its specific strengths and weaknesses. A comprehensive discussion about these measures, especially ROC has been done by Fawcett [39].

2.1.7. Retrieval Frameworks

All technologies described so far are different building blocks for a CBIR system. Except for the optional relevance feedback, each part is required. Research in the CBIR area often requires a prototype implementation and thorough testing of the outcome. Instead of building a search engine from scratch each time, the application of an existing framework should be considered.

An early solution was the Virage search engine [8, 50]. It already allowed to extend the basic system by simple or complex primitives. MetaSEEk [9] is a meta search engine, supporting the systems VisualSEEk [125], WebSEEk [23], QBIC [85] and Virage.

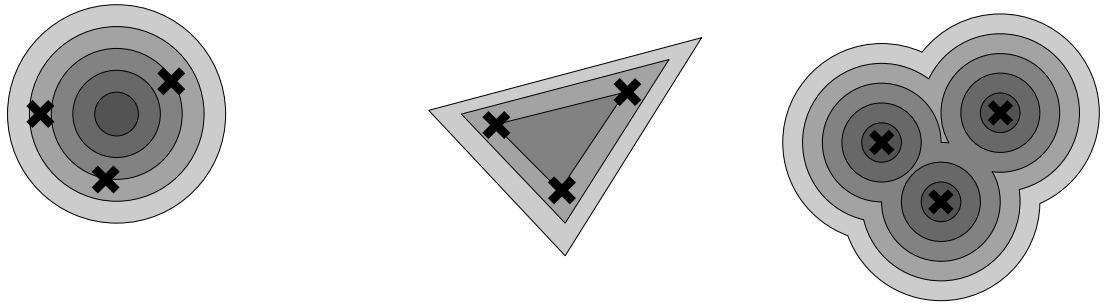
The Viper retrieval system [78] from the University of Geneva is based on the open

GNU Image Finding Tool (GIFT) framework. It uses altered techniques from traditional information retrieval. The idea is to merge several sources of information for the retrieval itself, such as textual as well as visual features. Currently the system is being replaced by a new development, called "ComMon SensE: Cross-Modal Search Engine".

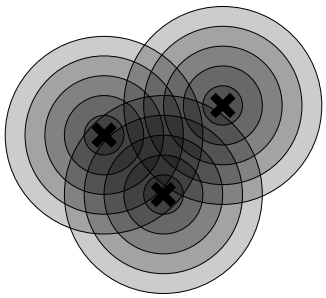
Joshi et al. [63] describe the recent, scalable web architecture PARAgrab for image retrieval, able to handle CBIR and keywords as well as interactive image tagging. It is part of a research group supervised by James Z. Wang at the Carnegie Mellon University, who also developed SIMPLIcity [138] and other CBIR related research.

2.1.7.1. Merging/Fusion

When merging multiple sub queries into a single one, there are several possibilities to do so. "Multi-modal fusion" is a recent research area emerging from the needs of modern retrieval systems.



(a) weighted average [65] (b) convex, clustering [65] (c) concave, disjunctive [65]



(d) concave, conjunctive

Figure 2.4.: Query shape

Kim and Chung [65] distinguish three kinds of merging multiple query points in a

single feature space for relevance feedback (figure 2.4). In fig. 2.4(d), the model used by the author of this thesis is visualized. It can be argued that all given query points are within the desired cluster. The search engine must now decide how the final similarity should be calculated. In the simplest case, all query points are averaged in the feature space to get a new query point. The generation of convex or concave similarity shapes is more complex, but provides advantages if the cluster has no circular borders. Especially the first two methods require the use of a simple distance measure. Otherwise the calculations may become overly complex.

The Rocchio's formula [105] is based on the vector space model and represents the query type in figure 2.4(a):

$$Q_1 = Q_0 + \beta \sum_{i=1}^{n_1} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{n_2} \frac{S_i}{n_2} \tag{2.1}$$

where Q_0 is the original query, R_i is the feature vector for the relevant document i and n_1 the number of relevant documents. This formula also considers non-relevant documents S_i by subtracting them from the query. The importance of the two sums can be justified by the weights β and γ [57]. A second formula for query point movement by Porkaew et al. [98] would be:

$$C[j] = \frac{\sum_{i=1}^{n} w_i E_i[j]}{\sum_{i=1}^{n} w_i} \tag{2.2}$$

where E_1 to E_n are the n query objects/feature vectors and w_1 to w_n are the corresponding weights. C denotes the resulting centroid to be used as the next query. Parameter j stands for a single dimension of the feature vector. The second approach depicted in figure 2.4(b) assumes, that all relevant documents lie close to each other in the feature space. The query points are clustered to a maximum of N clusters. For each cluster, the point closest to the centroid is used and weighted according to the cluster size. These form the next multi point query $M = \langle n, P, W, D \rangle$, where P is the set of points, W the corresponding weights and D is the distance function between two points. In this case — also proposed by Porkaew et al. [98] — the distance between M and any point x is:

$$D(M,x) = \sum_{i=1}^{n} w_i D(P_i,x) \tag{2.3}$$

The third approach (fig. 2.4(c)) is a multi point query, where each query point is taken as a separate query and the final result is merged from the sub results. This could be done by applying weighted sums, fuzzy sets or else. An example is the approach by Fagin [37], who interprets results as *graded sets*. They are basically lists sorted by similarity and set characteristics. He proposes to apply the basic Fuzzy rules defined by Zadeh [147]:

- Conjunction:
  $\mu_{A\wedge B}(x) = min\{\mu_A(x), \mu_B(x)\}$ (AND)
- Disjunction:
  $\mu_{A\vee B}(x) = max\{\mu_A(x), \mu_B(x)\}$ (OR)
- Negation:
  $\mu_{\neg A}(x) = 1 - \mu_A(x)$ (NOT)

Using the disjunction would directly produce the concave shape depicted in fig. 2.4(c).

In general, the merging approach which is most suitable for a given use case cannot be decided. This highly depends on the nature of the implemented features. Systems designed for experts should offer a choice, but for occasional use a decision has to be made by the administrator or the system itself.

This problem applies to a single feature with multiple query points and also to multiple disjunct features for a single query image. Further a more complex combination of these cases can be imagined. The proposed query language [95] offers the ability to create and use those queries. Though, it does not define the way of merging. A fourth merging alternative is the weighted sum of sub results (fig. 2.4(d)), where the similarities for all query points are added up and normalized [93]. Another, non-linear fusion approach called "super-kernel fusion" has been presented by Wu et al. [144].

2.1.7.2. Communication Protocols

It is remarkable that even after several years the existing prototypes are not yet established in daily life. Probably one reason is the diversity of approaches and the lack of interoperability. Recently emerged communication protocols could be the key to link many smaller systems together.

A communication protocol for this purpose, called Multimedia Retrieval Markup Language (MRML), is proposed by Müller et al. [81]. It is based on XML, has a formal specification and is already in use. The most recent proposal is version 2.0.

Currently, a de-facto standard for textual retrieval is emerging and gaining popularity. The Open Search Interface [1] is developed by *A9.com*. It wraps up several XML formats in order to allow federated search across the web. The whole standard is based on the assumption, that specialized engines are best suited to certain domains. Hence, a unified communication between clients and search engines is meant to be the best approach.

2.1.8. Categorization

Automatic image categorization — letting a machine determine, which semantic category an image belongs to — always involves some kind of more or less sophisticated machine learning. Common techniques are Artificial Neural Networks (ANNs) [33], Support Vector Machines (SVMs) [54, 141, 150] and Self Organizing Feature Maps (SOFMs) [22] or Decision Trees (DTs) [114]. Besides other techniques, there are two main approaches for machine learning, supervised and unsupervised.

Unsupervised techniques have no real need of training sets and manual annotation. They take a given set of input and try to find a describing model. A common way of doing this is clustering the data where no additional information is needed. The supervised techniques have in common that they are optimizing certain parameters to satisfy a training set with respect to certain classes as good as possible. A given input should lead to a defined output. An intermediate discipline is the semi-supervised learning, where a labelled training set is provided to learn a set of samples. This approach is a trade-off between the effort of annotating large amounts of data and the lack of semantic information.

In this investigation it is attempted to automatically build up descriptors based on a multi-feature query language. It is essential to provide at least a minimal ground truth, if certain concepts are to be learned in conjunction with a keyword. It can either be manually prepared for this purpose or has to be derived from the information context. This feeds essential information into the learning algorithm to optimize the descriptors. Thus, a supervised or semi-supervised approach is required if high-level semantics are to be learned. Semi-supervised learning has been successfully applied in image retrieval software, such as the "Multimedia Analysis and Retrieval System" [126] by IBM. The aspect of semi-supervised learning is further expanded in section 6.6.

2.2. Own Preliminary Work

This section summarizes related work of the author prior to the PhD itself, mostly contained within the master thesis [90].

2.2.1. Query Language

Prior to the PhD work, a CBIR query language has been developed by the author of this thesis [96]. It is based on the Lucene Query Parser [6] which defines a common language for full text search. The language allows queries similar to those used in traditional search engines and the parser is generated by JavaCC. This approach tries to merge key design principles of different languages. Like in OQUEL [134], queries are kept as simple and natural as possible. Yet, to provide a high machine readability, a strict grammar like in SQL is defined. The main extensions to provide CBIR functionality are:

fuzzy related operators and a *nested two-layer grammar*. The *boost* parameter for terms in the Lucene parser [6] has been extended in the master thesis of the author to multiple *TermParams* allowing additional control of fuzzy result sets [90].

To provide a high extensibility the grammar is split into two different layers. The basic layer (see 2.2.2) is parsed and interpreted by the search engine directly. This part of the grammar is predefined and fixed. Users may specify which meta information should be searched for by simply using fields. Typically these are fields like *title*, *content*, *author*, *creationdate*. Images hold other fields than normal text documents, typically Exchangeable image file format (Exif) and IPTC information. In the near future, this information may be replaced by the XML based XMP [104]. Additionally, a CBIR environment provides one or multiple feature vectors holding low-level information about the pixels. These feature vectors can be added by plug-ins, each one having a unique identifier which is the field name for content based queries. The difficulty now lies in specifying how the query feature vector is entered. There are at least three different ways possible:

- ID of an image stored in the repository
- Uniform Resource Identifier (URI) of a query image
- specification of the feature vector itself

The simplest way is to use an existing image for a query (*query-by-example*). Images already in the repository have the prepared feature vector available. Specifying the URI

of an image requires the engine to load the image and to extract the feature vector. The most advanced and complicated way is to let the user specify a feature vector in detail.

As a custom feature vector may contain any kind of proprietary data, offering an all-embracing language is not possible. Thus a second layer is added to the query language. A *Term* may contain the string *<FEATURE_START> [<FEATURE_CONTENT>] <FEATURE_END>*. The parenthesized part *<FEATURE_CONTENT>* is extracted by the search engine and passed to the responsible plug-in. The plug-in is fully responsible for parsing and interpreting this string to return the object representation of the feature vector.

2.2.2. Grammar

Below, the grammar of the query language in Extended Backus-Naur Form (EBNF).

```
Conjunction ::=  [ <AND> | <OR> ]

Modifiers ::= [ <PLUS> | <MINUS> | <NOT> ]

Query ::= ( Conjunction Modifiers Clause )*

Clause ::=
  [ LOOKAHEAD(2)
  ( <TERM> <COLON> | <STAR> <COLON> )
  ]
  ( Term | <LPAREN> Query <RPAREN> [TermParams]
  )

Term ::=
  (
     ( <TERM>  | <STAR> | <PREFIXTERM> |
       <WILDTERM> | <NUMBER>  | <URI> )
     [ <FUZZY_SLOP> ]
     [ TermParams [ <FUZZY_SLOP> ] ]
     | ( <RANGEIN_START>
         ( <RANGEIN_GOOP>|<RANGEIN_QUOTED> )
       [ <RANGEIN_TO> ]
         ( <RANGEIN_GOOP>|<RANGEIN_QUOTED> )
         <RANGEIN_END> )
       [ TermParams ]
     | ( <RANGEEX_START>
         ( <RANGEEX_GOOP>|<RANGEEX_QUOTED> )
```

```
       [ <RANGEEX_TO> ]
         ( <RANGEEX_GOOP>|<RANGEEX_QUOTED> )
         <RANGEEX_END> )
        [ TermParams ]
     |
       ( <FEATURE_START>
       [ <FEATURE_CONTENT> ]
        <FEATURE_END> )
        [ TermParams ]
     | <QUOTED>
       [<FUZZY_SLOP> ]
       [ TermParams ]
  )

TermParams ::=
    (
     <CARAT> boost (
      ([ <HASH> maxCount ]  [ <AT> threshold ])
    | ([ <AT> threshold ]  [ <HASH> maxCount ])
    )

    |   <HASH> maxCount (
      ([ <CARAT> boost ]  [ <AT> threshold ])
    | ([ <AT> threshold ]  [ <CARAT> boost ])
    )

    |   <AT> threshold (
      ([ <CARAT> boost ]  [ <HASH> maxCount ])
    | ([ <HASH> maxCount ]  [ <CARAT> boost ])
    )
)
```

2.2.3. Plug-Ins

The plug-in concept of the retrieval framework described in [90] allows the definition of any new feature. To make such a plug-in available in this language, only a few requirements need to be met. The plug-in needs an identifier which is automatically used as a term field. With this information it is already possible to formulate queries containing an example image (either by internal id or URI). The tricky part is to develop a syntax for user defined feature vector information embedded in a query. As features can be arbitrarily complex, each plug-in should use a simple default language to describe the

contents of a feature vector. Otherwise the embedded data string of a query is forwarded directly to the feature plug-in where it needs to be converted into a valid feature object.

2.3. Identification of Problems

This section highlights major problems in the research fields analyzed in section 2.1. This collection of issues is used to identify the aims presented in section 2.4.

The whole MIR area has to deal with fuzziness and uncertainty. Both approaches — either using primitives or semantics — are not yet suitable to create satisfying results on large data bases. A retrieval system based on low-level information is too imprecise, as a deeper understanding of the content is missing. A semantic based approach requires a lot of human annotation effort to be correct. Semantics created automatically also rely basically on low-level analysis and some human annotation.

A great deal of the problems in MIR seem to be caused by an imperfect human-machine interaction, missing machine intelligence and a lack of standards.

Most papers in section 2.1 indicate that human beings are still better in generic object recognition than machines. Thus, the information which is already available should be used in the best way possible.

2.3.1. Browsing

For browsing, the search engine must present an initial overview of the contents. In pure CBIR, this is usually done by generating an initial image subset of the database. This can either happen at random or by a specific query from the user. Based on the user input, a more specific overview of probably related items must be generated.

In CBIR, often a number of thumbnail images is presented to the user [50, 85]. It is even possible to present a quite large number of images. Unlike text, image content can often be perceived at a glance. The user may now scan the results for potential matches. It is still an open question, as to how the results should be arranged. Several different ways [26, 69, 71] have been already proposed, each one with certain advantages and drawbacks. Sorting these hits by categories or visual similarity seems to be beneficial in many cases. To find eye-catching images, a random distribution seems to be even more useful, while the user does not necessarily need to know the arranging method [106, 108]. The challenge is to find a convenient solution.

2.3.2. Query Language

Independent from the user interface, each search engine needs to support queries. Usually they define a string representation covering the whole set of parameters. Internally the string is parsed and probably optimized before it is executed. The application area has a strong influence on each individual grammar. Image retrieval systems often provide a visual interface to compose queries.

Currently, the CBIR systems available mostly seem to use a proprietary query language, specifically adapted to each particular implementation. Several attempts have been made (e.g. FOQL [84], OQUEL [134]), but none of the proposed languages was highly successful in CBIR yet. The main reason for that seems to be the immense variability of requirements to such a language, as each system focuses on a certain field. In the current research the most important aspects about querying are:

- How to formulate a meaningful query?
- How to support as many useful queries as possible?
- How to support the user in searching/browsing the results?

Another issue is the possible variety of user expertise. People working at workplaces with an immense amount of data, such as travel agencies or libraries are used to deal with complex query strings. But these queries are only special to a single application area and require a good deal of training.

2.3.3. Relevance Feedback in Search Results

Relevance feedback requires user interaction to work properly. The system behaviour is adapted to the user. For this reason the newly provided information must be more correct than the current situation. Otherwise the future results cannot be improved in quality. Further, the user is probably forced to do additional work which actually slows down a current retrieval session. In such a case, the search engine taxes the user's patience.

2.3.4. Features and Similarity Measures

There are two important characteristics of a feature in retrieval scenarios. The feature vector requires to be highly descriptive for a given document and there needs to be a

similarity measure for each two feature vectors of the same kind. A feature vector usually captures one certain document feature in a highly condensed format. This format does not allow a reconstruction of the original data, but it is describing a certain part of it.

The multi dimensional nature of many feature vectors turns out to be very difficult to define a generally valid similarity measure. Each feature has its own structure and every dimension in the structure may reflect specific semantics, which cannot be easily captured. In the case of simple mathematical vectors, a default distance measure could be applied. Often a low dimensional Minkowski/L_m distance is used. Very popular measures are the Euclidean distance (L_2) and the Manhattan distance(L_1) [90]. Standard index structures like multi dimensional trees such as the R-Tree [51] could be applied.

In some cases, the vector representation is too coarse to capture all aspects of a feature. A more complex distance calculation between two features could be based on another feature model. The similarity function does not even have to be inverse.

An example is a set of keywords attached to each image interpreted as image feature. It has been implemented by Pein [89] in an earlier CBIR prototype. The image to be retrieved is represented by the set $I = \{a, b, c, d\}$ and the search query is $Q = \{a, c\}$

The similarity between these sets could be defined as $s_{IQ} = \frac{|I \cap Q|}{|Q|}$, which is the relative occurrence of the query terms in the image feature. In this example, the resulting similarity would be $s_{IQ} = \frac{\{a,c\}}{\{a,c\}} = \frac{2}{2} = 1$. Swapping image and query set, the result is different: $s_{QI} = \frac{|Q \cap I|}{|I|} = \frac{\{a,c\}}{\{a,b,c,d\}} = \frac{2}{4} = 0.5$.

From a users point of view, this behaviour appears consistent. By specifying several keywords, one would expect, that the highest ranked results would contain all of them. Consequently, the relative amount of matches should be taken into account.

In these cases none of the basic feature vector theories can be applied without care. This disqualifies these features for many standard optimizations. But sometimes other ways of feature specific, but more efficient indexing can be found. E.g. for keywords, the reverse index is a highly efficient structure.

2.3.5. Annotation

The annotation problem is the foundation of all object recognition effort. Single images and small repositories can be easily annotated by humans whereas a growing amount of information cannot be handled any more by a single person. Due to the fact that object recognition is still an open problem, it is necessary to find a practicable workaround for the time being.

2.3.6. Evaluation of Retrieval Systems

It is claimed that benchmarking in CBIR is a very controversial topic [139]. Due to the immense fan out of use cases and application areas, the aims of all the systems vary. It is a common opinion that synthetic benchmarks could kill innovation and causing new systems to be focused on solving the pre-defined tasks. For this reason, generic benchmarks should only apply to already established and small application areas. Judging the whole of research by a set of standardized benchmarking criteria would probably slow down innovation.

2.3.7. Retrieval Frameworks

It is obvious that until now no CBIR framework has made a final breakthrough. This might also be caused by the diversity of objectives.

An important problem is the lack of a generic indexing structure for features. Possible approaches are not implemented within the boundaries of the thesis but are going to be considered in the future. Examples are used in the prototypes by Joshi et al. [63] and Quack et al. [100].

Which merging approach for sub-queries is most suitable for a given use case, cannot be decided generally. This highly depends on the nature of the implemented features. Systems designed for experts should offer a choice, but for occasional use a decision has to be made by the administrator or the system itself. The proposed query language [95] offers the ability to create and use complex Boolean queries. However, it does not define the way of merging.

2.3.8. Categorization

In the field of image recognition, several case specific problems arise. A major problem is to find suitable samples representing a certain object. For optimal learning, the object has to be totally isolated from the image background. Otherwise, the algorithm may also learn to recognize irrelevant noise instead of the relevant object only.

Further, most images do contain more than a single object. In that case, it is important to distinguish them and inform the learning algorithm which region represents which object. Overlaps, irregular shapes and high-contrast patterns increase the difficulty of automatically clustering images.

It is also important to keep in mind, that multiple samples of a single concept could differ significantly from each other. Using low-level descriptors and simply calculating

a single average value or centroid is not sufficient in many cases. Then a "divide-and-conquer" approach for detailed modelling is required, as investigated by Wu and Nevatia [143]. They propose an iterative algorithm to subsequently add specific classifiers to a classification tree. If the category to be learned cannot be matched by a simple classifier, the training samples are spilt into smaller and more specific clusters. As a result, the classifier complexity grows with each single split and methods for joining them afterwards are required.

2.4. Aims and Objectives

The recent overview provided by Datta et al. [30] concludes with the statement, that the research focused more in *systems*, *feature extraction* and *relevance feedback* than in application-oriented aspects such as *interface*, *visualisation*, *scalability* and *evaluation*. Thus it seems desirable to improve research in these areas.

According to Lew et al. [70] there are several recent research topics trying to "bridge the semantic gap". In human-centred computing the system tries to satisfy the user while keeping the interface easily understandable. One possible approach is to apply machine learning to connect low-level features to high-level semantic concepts

The currently available technology can be applied to a CBIR system. This thesis examines a possible way of reducing the semantic gap. It focuses on how to link low-level features to high-level semantics. Key technologies to achieve this aim are:

- a user interface allowing guided browsing of an image repository
- a query language that can handle arbitrary features
- a core retrieval system supporting multiple feature vector types simultaneously
- a decision tree based learning algorithm to build category-related queries

User Interface The user interface to be developed needs to address the problems stated in the former sections.

Section 2.3.1 indicates that the process of retrieving images depends on the users needs and the particular use-case. Thus, the user interface should offer guidance for inexperienced users and detailed control for experts.

Query Language In section 2.3.2 the query language is identified to be the main connection between user interface and retrieval system. It is attempted to determine the crucial parts of a versatile language and to define a grammar covering them. Further, this language should be easily convertible between human-readable, XML and machine representations. The recently developed query language [95] is used as a foundation and is examind in detail.

Core System The core system relies on mathematical models rather than human interaction. The back end of the project deals with the problems arising on pixel-level and fuzzy definitions. Having no universal feature covering all requirements in a single feature, it seems to be necessary to keep on researching new feature models and similarity measures. Also, these different features need to be mergeable.

A CBIR framework should wrap up all of the aims above into a fully functional system. The optimal system would be flexible enough to be set up in many different use-cases.

Categorization To reduce the need of building up a suitable and annotated image repository, some prepared image collections with category annotation are used in this investigation. These images usually represent a single concept and the object of interest is either located in the centre or fills almost the complete image. This approach largely avoids the trouble of image segmentation.

A couple of previously implemented low-level features are then used as a basis for classification. Having various features available for optimization allows for determining the best ones for each single object class. The use of a query language with Boolean operators allows the learning algorithm to specify multiple independent clusters at the same time. Each single cluster may vary in its boundaries and the feature used.

The resulting query descriptors should be describing a given concept as well as possible. As with most learning algorithms, the problems of over-fitting and unpredictable run times have to be addressed. Thus, suitable heuristics for this learning scenario need to be developed.

Image collections containing several images for a single concept are used. Two candidates are the Caltech-101 [40] and the ETH-80 [67] repository. Both contain between 31 and 800 samples for each contained category and the main object is centred in the images. As all feature vectors used within this thesis are global features and no segmentation is applied, it is important for the image sets to fulfil the following preconditions:

1. Each image represents a single object/concept

2. The object/concept to be learned fills most of the image area, is centred and completely visible

3. The ETH-80 collection provides additional mask files which define the relevant segment

Finally, the developed system is evaluated. This is done by comparing the query descriptors learnt in the categorization stage to an existing approach.

Chapter 3.

Methods Employed

This chapter points out certain technologies that will be applied for a CBIR system. A prototypical retrieval system is developed in order to evaluate the research hypothesis (section 1.1).

Typically, a CBIR system consists of *user interface*, *query-processing-module* and the *image database* (figure 3.1). The user interface is probably the most important part of a search engine. An engine producing perfect results is of no value, if it cannot be handled. It is necessary to reduce the *perceived* complexity for the user. The approach is to provide graphical guidance while hiding the complex background theories as much as possible. This is especially useful for inexperienced users. Experts have usually far less difficulties in understanding and using complex interfaces. A real life example is the comparison between graphical user interfaces and console applications. Programs started from a console offer several parameters for detailed configuration. This requires knowledge of the program name and the parameter syntax. Their graphical representations are in many cases much easier to handle, but they offer less parameters and actions. For this reason, expert users should always be able to have a lower level access than the beginner.

3.1. Browsing

A basic browsing loop during retrieval is shown in figure 3.2. The user starts with an initial query, waits for the results and then checks the results. In the best case, the desired content is already displayed on a prominent position on the screen. If not, the user may navigate through more results that are not yet visible on the screen. To improve the results, the initial query may be refined or even rewritten completely to initiate a new search.

This work flow contains all the important stages that need to be presented to the

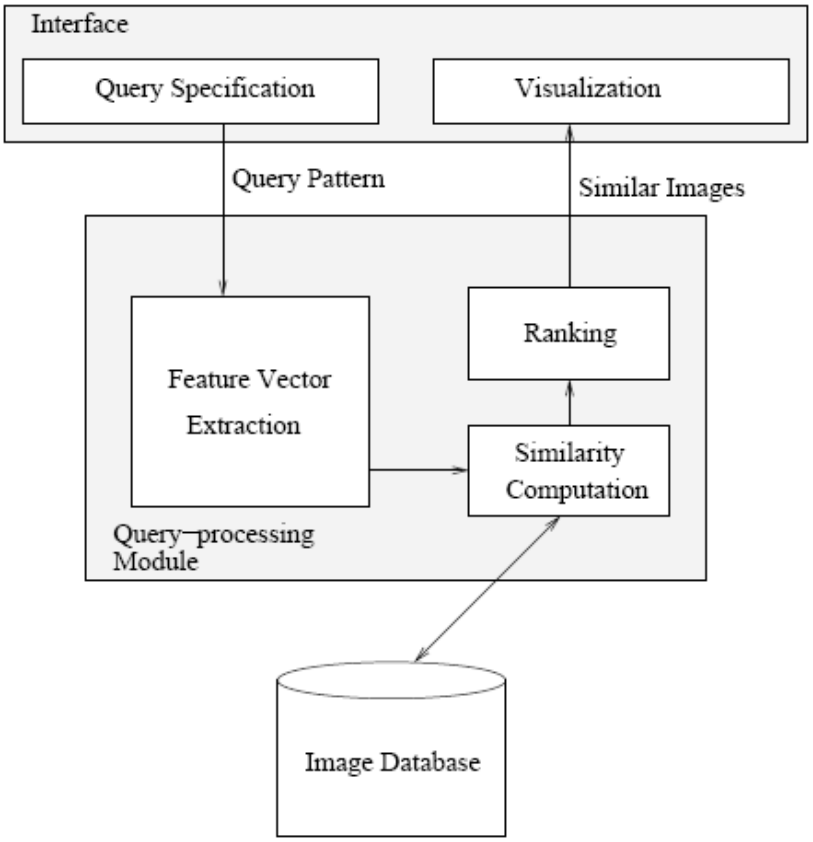


Figure 3.1.: Ranking in a Typical CBIR-System [133]

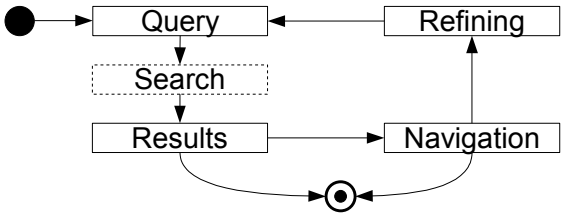


Figure 3.2.: Retrieval-Workflow [94]

searcher. Each stage requires some methodology and techniques applied in the retrieval system.

For efficient browsing, the user needs to see some content of the available repository. This could be a structure of available categories, the results of a preceding search or other information.

The developed system does only provide a one dimensional list of results, as the aspect of browsing cannot be investigated in depth within this thesis. This list is always generated by a query which either generates a set of random results or represents a normal retrieval.

3.2. Query Language

In section 2.3.2, a set of important aspects for the design of a query language is mentioned. The requirements for the query language applied in this thesis are derived from these aspects. Converting the users mental model into something machine readable inevitably causes a loss of information. The task of a query language is to minimize this loss. Based on this assumption, the requirements are composed and presented below.

3.2.1. Query Language Requirements

Which aspects does a query language for a CBIR system need to cover?

- Boolean Queries
- Nested Queries
- Feature Vector Queries
- Keyword, Tag Queries
- Querying Concepts ("discrete"/fuzzy):
  - Weights and Preferences
  - Filters
  - Temporal Aspects [53]
- Technology:
  - Query-By-Example (existing image/upload tool)

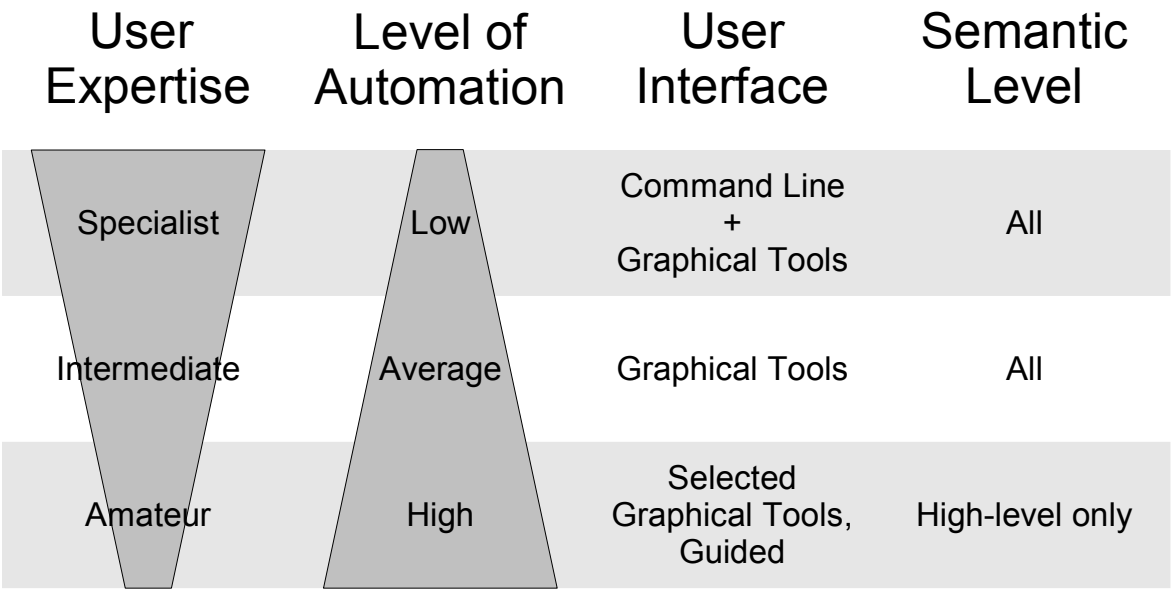


Figure 3.3.: User Experience Levels

- Query-By-Sketch (drawing tool)
- Query-By-Feature (input tool)

As the level of user expertise may differ, the interface needs to provide multiple levels of complexity. Thus, is seems to be necessary to provide several user interface levels (fig. 3.3). The users can be split into three main categories:

specialist users Basic query language with all possible parameters and constructs, string based "console"

intermediate users Complex visual language constructs to compose queries close to the abilities of the underlying language, some tool support

amateur user Simplified visual language, containing only the essentials and much guidance/tools

Beginning with the lowest level, more and more complexity of the underlying query language will be hidden. This usually leads to better usability as well as less freedom of manipulation. In fact, the whole retrieval system has to rely on a sound and powerful query language.

On higher abstraction levels, the user may be guided through a use case dependent work flow. The system could decide in the background, which low-level features suit the

Table 3.1.: Query Composing

Type	**Advantages**	**Disadvantages**
Query String	highly flexible	must be learned complex features tedious
Query-By-Example	simple	seed image by upload or other query
Query-By-Sketch	no image required	relies on user's painting skills requires canvas
Query-By-Feature	no image required	user needs basic feature understanding
Random	trivial rough overview	trivial
Browsing by meta data	good overview directed search	requires structured/ annotated content

users needs and how they should be weighted. This functionality needs either to be set up by an experienced administrator, or by a learning algorithm. These could be realized with techniques like use neural or Bayesian networks. The training is usually performed by relevance feedback or dedicated training sets [16, 33, 97, 109, 120]. In section 3.7, a learning approach is presented.

3.2.2. Query Composing

Especially amateurs expect to see a well designed interface. The learning curve ought to incline very gently. Otherwise the user might be discouraged before even entering the first query. Being all about images, the CBIR scenario virtually suggests itself a visual query interface. Several query types related to CBIR are listed in table 3.1. Though the most flexible approach, plain query strings are only of use, if they are easy-to-use or for experts. QBE is still feasible with URLs and Query-By-Feature (QBF) could also be represented by a serialized representation. The Query-By-Sketch (QBS) finally requires a graphical interface. To complete the collection of query types, the random and meta data queries are also important. Without any random capabilities, QBE could not work properly. No one can expect that the user has suitable example images at hand to be uploaded.

Composing queries by drawing them with objects from a toolbox directly, shows the user the most important possibilities. Without reading any documentation, the user can see useful constructs and try them out directly. Further, a visual query is not based on

natural language. This keeps specifications clear and allows high complexity without any parsing errors.

The main task is to reflect all essentials of the underlying query grammar. Special nuances should always be possible by falling back to a basic query string. With a sound query grammar, the mapping from query objects to visual components and back is not a difficult task. Even the generation of basic input forms automatically created by the use of formal field definitions can be applied.

In addition, the visual query language should at least provide support for Boolean queries, image features and some parameters. Especially the features require attention, because entering them manually can be a very tedious work. There should be a visual tool to create and alter features. Otherwise, it can only be used for QBE. Considering the fuzzy nature of CBIR, these features need to understand range queries and wild cards.

Having graph like queries, it is of importance, how the objects are spatially arranged. The amount of overlaps and crossing joints should be low. Of course, incremental query building should be also supported, allowing to integrate newly found information into the query. This way, the query might grow very complex. In this case, sub-queries should be visually collapsed to hide details of an internal sub-query.

3.2.3. Query Language Principles

According to the problems mentioned in section 2.3.2, the query language should follow these basic principles:

- keep it simple
- keep flexible
- keep it parseable
- avoid ambiguities
- stick to a reasonably sound mathematical grammar

This ensures both a reasonably understandable language as well as a sound theoretical background. The basis chosen for this is the language proposed earlier [95]. Designed to be a flexible CBIR language, it provides all the functionality needed for the next steps. It is also easily convertible into an object-oriented format. These objects will be

the nodes of the query graph. These nodes can hold and display all the information of the query. That is a visual representation of query images or a given feature as well as additional parameters.

3.3. Relevance Feedback

Relevance feedback gives the user some control to influence the search results. Usually the user can judge the quality of a result set and give the system hints to improve future retrieval sessions. The feedback mechanism is always closely related to the user interface, either to allow direct input or to analyse the users behaviour in the background.

Short-term relevance feedback can be implemented by simply altering the query in the background. If the user dislikes certain images, they are added as a negative example with a certain threshold to remove a couple of images from the result set.

Long-term approaches require much more attention, as the index itself is altered. The impact of wrong information needs to remain low and the weight of positive must be higher than negative ones Müller et al. [77].

The integration of relevance feedback in the developed system is not considered due to resource constraints. Nevertheless, a short-term relevance feedback mechanism is technically available on a low level. The boolean structure of the query language itself offers a couple of parameters to alter results.

3.4. Features and Similarity Measures

As the application areas for CBIR are multifaceted, it seems to be necessary to use multiple feature vectors to describe similarity.

3.4.1. Feature Evaluation

The feature vector evaluation can be measuring several parameters of each feature vector. An exemplary evaluation dealing with the impact of the query image size is described in section 6.1. Due to restricted resources, further measurements have not been prepared within this thesis. The case study follows a common pattern that can be applied to other parameters as well.

This pattern (algorithm 1) uses multiple modifications of the original image as query. This is done by scaling it down in several steps to square images of defined width and

height. The loss in accuracy is measured by the decline of similarity compared to the original feature vector and also by determining the rank of the original image in the result of the altered query.

Algorithm 1 $evaluateFeatures()$

```
1: for all i ∈ I, f ∈ F do
2:     Scales ← {160, 80, 40, 20}
3:     for all scale ∈ Scales do
4:         i_scaled ← downscale(i, scale)
5:         f(i_scaled)
6:         s ← calculateSimilarity(f(i_scaled), f(i))
7:         rank ← pos(i, getRanking(f(i_scaled)))
8:     end for
9: end for
```

3.4.2. Feature Normalization

It is obvious that every feature vector developed has different characteristics. Sections 2.1.4 and 2.3.4 discuss some examples, where data format and the maths model are of various kinds. Also, calculating the similarity of two feature vectors highly differs. Earlier tests with implemented feature vectors indicated that the similarity values of two different feature vector models cannot be compared directly. While one feature vector model may usually rank the first 100 hits with a similarity of 0.99, another feature vector model may hardly ever return such a high value except from identical images [90]. Thus, it seems to be beneficial to provide a normalization mechanism for feature vector plug-ins.

The challenge is to determine a useful normalization function that can be applied to every similarity calculation to make the results comparable with other, completely unrelated features. To do so, the behaviour of the similarity distribution of each feature vector is required.

3.4.2.1. Similarity Profiles

An empirical approach is used in order to determine the individual characteristics of a given feature vector model. This information is needed to normalize the similarity distribution of the model.

Every feature is tested against the same image dataset. Every image is used as a query to generate a full result set of the complete repository. These results are sorted

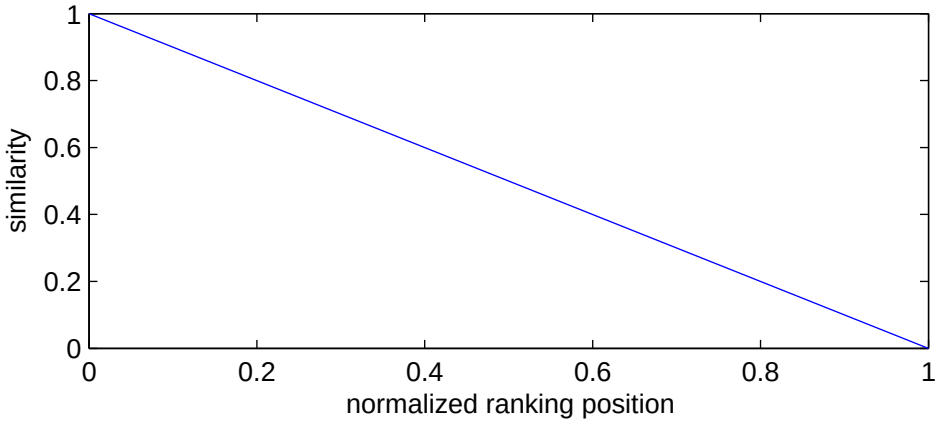


Figure 3.4.: Optimal Similarity Profile

by decreasing similarity. For each ranking position, the similarities of all search runs are accumulated, i.e. the average, the highest and lowest similarity.

The resulting profile consists of three monotonic decreasing discrete functions with $rank \in [1, |I|]$ and $average(rank) \in [0.0, 1.0]; min(rank) \in [0.0, 1.0]; max(rank) \in [0.0, 1.0]$. These functions can be easily compared against each other and are the starting point for a normalization function. Naturally, these profiles depend on the image dataset used and may look significantly different with another repository. In section 6.2, two image collections, mainly consisting of photographs are used.

3.4.2.2. Determining a Normalization Function

Aim of the normalization is to achieve a steadily decreasing profile. The first ranked image should have a similarity of 1.0 whereas the last ranked image should be at 0.0. In between, the profile should decrease steadily. In terms of a function, the optimal normalized profile would be:

$$s(x) = 1 - x; x = \frac{\text{rank}}{|I|} \tag{3.1}$$

where rank is the ranking position of an image, $|I|$ is the size of the repository and s is the similarity of the image to the query. The optimal profile from eq. (3.1) is visualized in fig. 3.4.

A normalization function requires to have the following characteristics:

- every value $x \in [0.0, 1.0]$ must be mappable to an $f(x) \in [0.0, 1.0]$

- $f(0.0) = 0.0, f(1.0) = 1.0$
- $f(x)$ must be monotonic ascending
- determinable by an algorithm
- fast calculation

The retrieval system requires every feature vector plug-in to return only similarity values in the range $[0.0, 1.0]$ (section 3.6.2). Thus, every normalized value must also lie in between these boundaries. The end points of this range are fixed and must not be changed. Further, the original ranking order must also be maintained. This demands for a monotonic ascending function, preferably strict. To allow the system to adapt to changing environments, the normalization function should be automatically generated by an algorithm. Finally, the normalization should be fast to compute as it needs to be performed for every result in every search request.

In order to generate a normalization function, a set of data points $\sum_{i=1}^{n} P_i(x_i, y_i)$ for interpolation can be extracted from the profile. These data points should be chosen carefully to ensure a high quality. If too few points are chosen, the interpolated function may become inaccurate. Too many points may unnecessarily slow down the calculations.

For interpolation between each two points $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$ in the normalization function, the common linear model $y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}$ is chosen. It is fast and ensures keeping the monotonic characteristics of the profile. Polynomial or spline interpolation may cause overshooting of the curve, thus changing the ranking order. Also, a perfectly smooth normalization function is not mandatory, as a very high accuracy is not demanded and the functions does not need to be differentiated.

3.5. Evaluation of Retrieval Systems

The current effort put in the development of benchmarks is immense. Whole conferences are determined to apply benchmarks to restricted application areas. It is almost impossible to find a globally useful benchmark.

Nevertheless, default metrics such as precision/recall should always be used when testing the performance of a system. Also it is in the responsibility of each researcher to pick an appropriate data base for testing. Each available benchmark has its right to exist, but one must carefully choose the right ones.

In this thesis, two existing databases (i.e. ETH-80 and Caltech-101) are used for evaluation. The basic metrics used are precision π and recall ρ that are defined as follows according to van Rijsbergen [136]:

$$\pi = \frac{tp}{tp + fp} = \frac{|R \cap retrieved|}{|retrieved|} \tag{3.2}$$

$$\rho = \frac{tp}{tp + fn} = \frac{|R \cap retrieved|}{|R|} \tag{3.3}$$

where tp stands for the *true positives*, fp for the *false positives* and fn for the *false negatives*. The *true negatives* tn are not used in these equations (see appendix A). In order to capture both precision π and recall ρ in a single value, the F-Measure or the effectivity measure by van Rijsbergen [136] is applied:

$$F_\beta = (1 + \beta^2) * \frac{\pi * \rho}{\beta^2 * \pi + \rho} E = 1 - \frac{1}{\alpha(\frac{1}{\pi}) + (1 - \alpha)\frac{1}{\rho}} \tag{3.4}$$

where β and α are weights to modify the balance between precision and recall. In section 6.6, the results of the system are also compared to a reference learning approach.

3.6. Framework

The retrieval engine is supposed to be powerful enough to handle both textual and content-based queries. Primarily the engine needs to fulfil the requirements given by the user interface. Mainly the previous prototype [90] and the Lucene tool kit [6] were used.

3.6.1. Similarity Search

The main problem is the lack of a generic indexing structure. While small repositories can handle a linear scan, large ones require better solutions. Approaches like pre calculating clusters of similar images and the use of multidimensional trees are to be implemented. Examples are the prototypes by Joshi et al. [63] and Quack et al. [100]. In order to minimize potential side effects of index based retrieval, no indexing technique is applied. Instead, a full scan is performed during ranking.

3.6.2. Merging/Fusion

The main difficulty of combining sub results from a CBIR system is the fuzzy nature of the results. Some simple features with filtering character (e.g. keywords) deliver a rather clean set of hits. But it is essential to have a a fuzzy model for merging these with highly similarity based features. Those results are usually a sorted list [37, 103]. The approach by Fagin [37] interprets results as *graded sets*, which are lists sorted by similarity and set characteristics. He uses the basic rules defined by Zadeh [147].

The text retrieval concept of *boosting* single terms by any float value is adapted to the extended engine. Before merging sub results, the similarities are boosted as specified to shift the importance into the desired direction.

An additional acknowledgement to the fuzzy nature is the use of additional set operators to keep the results at a reasonable size. The *minimum similarity* is a value between 0.0 and 1.0 and forces the engine to drop all results below this similarity threshold. As the efficiency of the threshold highly depends on the available images and features, a *maximum size* parameter limits the result to the specified size.

In the current prototype, a query q (eqn. 3.5) is defined in the following way:

$$\begin{aligned} \mathrm{q} &= C \cup C^{+} \cup C^{-} && (3.5)\\ c &= \text{subquery or term} && (3.6)\\ C &= \{c | c \in \mathrm{q}\} && (3.7)\\ t &= \text{field/feature name and data } f(i) && (3.8)\\ f(i) &= \text{feature of image } i && (3.9) \end{aligned}$$

where c is a subquery or term and t is a single term that can be processed as a single retrieval request. The subsets C, C^{+} and C^{-} contain the clauses for $SHOULD\_HAVE$, $MUST\_HAVE$ and $MUST\_NOT\_HAVE$. The similarity is defined as:

$$\begin{aligned} s(f(i_{\mathrm{q}}), f(i_x)) &= \text{similarity of } i_{\mathrm{q}} \text{ to } i_x \text{ by } f && (3.10)\\ &= s_x \text{ (short hand)} \end{aligned}$$

where $s(f(i_{\mathrm{q}}), f(i_x))$ is the similarity between two images i_{q} and i_x. It is defined for each feature f independently. The result sets are defined as:

$$r(\mathrm{q}) = [r(C^+) \cap r(C)] \setminus r(C^-) \tag{3.11}$$

$$r(t) = \{(i_x, s_x) | i_x \in I \wedge s_x \in [0.0, 1.0]\} \tag{3.12}$$

$$r(c) = \begin{cases} r(\mathrm{q}) & \text{if } c \text{ is subquery} \\ r(t) & \text{if } c \text{ is term} \end{cases} \tag{3.13}$$

where the results $r(c_x)$ of each boolean clause c_x are a set of tuples (i_x, s_x). Each one contains a retrieved image i from the repository I and the similarity s to the query feature. These sub results are merged according to the unary operators of each clause:

$$r(C^+) = I \bigcap_{x=1}^{n} r(c_x^+) \tag{3.14}$$

$$r(C) = \bigcup_{x=1}^{n} r(c_x) \tag{3.15}$$

$$r(C^-) = \bigcup_{x=1}^{n} r(c_x^-) \tag{3.16}$$

First, the $MUST\_HAVE$ results $r(C^+)$ (eqn. 3.14) are intersected, which hopefully reduces the total search space for the $SHOULD\_HAVE$ clauses that generate $r(C)$ (eqn. 3.15). This possibility of optimization is the main reason why no established fuzzy concept as proposed by Fagin [37] or Zadeh [147] is currently used. Based on this subspace, the second part of sub results is collected. These are joined and obtain a new weighted relevance. Finally, the $MUST\_NOT\_HAVE$ results $r(C^-)$ (eqn. 3.16) are removed from the final results $r(\mathrm{q})$ [90].

3.7. Categorization

According to Schapire [113], classification tasks can be successful when using multiple "weak learning" algorithms. An important requirement for each weak learning algorithm is the ability of having at least a small classification advantage over random guessing. This section describes a learning approach published by the author of the thesis in more detail [91].

3.7.1. Unsupervised Learning

A totally unsupervised learning is not feasible for directly linking keywords to learned categories. Yet, it could be applied in order to build unlabelled clusters of similar images for each feature space. These clusters may be useful as an index for fast retrieval or to provide a quick overview of the repository [112]. They could also be manually annotated afterwards, if they are mainly describing a single concept.

Within the boundaries of this thesis, this type of learning is not applied.

3.7.2. Supervised Learning

The proposed theoretical model is based on the following assumptions which are essential to justify the decisions described below. Most of which are necessary simplifications to tackle the still unsolved problem of the "semantic gap" [61, 124].

1. High-level semantics can be described by a set of low-level features
2. Certain low-level features are efficient in separating a set of relevant images from irrelevant ones
3. A reasonably large number of mutually independent features must be available to choose from to capture specific types of similarity
4. A single semantic concept can be represented by highly dissimilar images or image features (i.e. there is not necessarily a single "best describing feature")

Following these assumptions, a feature based query language (e.g. [96]) can be used in machine learning. Each concept to be learned can be mapped to a query containing low-level features only. The learned query itself would be a set of (boolean) combined low-level characteristics extracted from a training set. The algorithm suggested in this thesis analyses the results of CBIR queries to build a classifier. It is similar to the Cluster Boosted Tree approach by Wu and Nevatia [143], which is based on the AdaBoost algorithm by Freund and Schapire [42]. In their algorithm, the training set is clustered according to the features before extracting the most suitable features for each sub cluster. A drawback of their approach is, that complex features may not be directly suitable for clustering, as the similarity measure is not necessarily identical to a multi-dimensional distance.

Capturing disjunctive sample sets of a single concept can be described with a disjunction (OR) operator. Samples of a semantic set are likely to have different low-level

features. But according to the theory of feature vector based retrieval, series of samples are expected to be organized in clusters. These clusters are bounded by the features used. Often, a specific feature can be further optimized by removing irrelevant information [99], making it more expressive. Yet, it should not be the task of a single low-level feature to capture all possible samples and merge them into a single "all-knowing" descriptor. Instead, a higher level descriptor should merge them into a single semantic concept. According to Zhang and Izquierdo [148], this usually cannot be done by simply merging low-level descriptors directly, because many low-level features show a non-linear behaviour.

Some low-level features are very efficient in filtering a certain concept from a given repository. But inevitably, using only a single feature, the filtered results may still be cluttered by several false positives. Further filtering may be required and two cases should be considered separately. If the false positives are very similar to each other, a negated term (NOT) could remove the unwanted parts from the result. Another approach to improve a filter is to specify further, unique features to the desired concept. This can be modeled by a conjunction (AND) to define boundaries in multiple feature dimensions. Rather than trying to create a single "multi-feature centroid", the proposed approach tries to find multiple relevant centroids, regardless of the features used.

The learning algorithm requires a training and an evaluation set. In the case of supervised learning, both sets are identical. For each image of a concept, the suitability as a representative query term or "centroid" is checked. This is determined by the ability of containing many other related images in the query result set. In other words, precision and recall should be as high as possible.

The best describing retrieval terms can then be further optimized by calculating a suitable similarity threshold. These optimized terms are then combined in a single query by the boolean "OR". If the low-level features applied are suitable to capture the relevant characteristics for the learned concept, the resulting query should be able to return a set of concept related images with only a few errors.

In some cases, a set of unrelated images may find its way into the results. This effect could be reduced by adding representative "AND" or "NOT" terms to a query. This term should not be closely related to an existing feature in the positive query part to stress the desired or to "cut out" the unwanted parts with a high accuracy.

The final classification query then needs to be tested against the remaining evaluation set. The main target is to find the optimal balance between maximum query length and retrieval accuracy. The more terms are used, the slower the final retrieval will be and

the risk of over fitting increases. The query can then be used to calculate the similarity of any unknown input image to the query-related class, which can also be interpreted as the probability of belonging to that particular class.

3.7.3. Semi-Supervised Learning

The transition from supervised learning to semi-supervised learning would essentially be to shrink the labelled training set to a small size. According to Smith [126], a reasonable size trade-off would be approximately 10%. An alternative way of testing is the leave-one-object-out cross validation. Such a test, leaving one of the objects of a category out has been published by Leibe and Schiele [68] using the ETH-80 dataset. A similar test is performed in section 6.6 for direct comparison.

The semi-supervised learning generally follows the same principles as the supervised approach section 3.7.2. The main difference is the change in the training set T and the evaluation set E. Semi-supervised learning uses disjunct sets generated from the known relevant images.

3.7.4. Definitions

The proposed learning algorithm uses several sets of images. A repository contains a set of images $i \in I$. In order to start a training session, the repository is split twice. The concept to be learned is represented by the set of relevant images R. The remaining images N are considered to be non-relevant. For later evaluation, R is again split into two sets: the training set T and the evaluation set E. The describing features are contained in the feature set F.

$$T \cup E = R \tag{3.17}$$

$$T = E \text{ (supervised)} \tag{3.18}$$

$$T \cap E = \{\} \text{ (semi-supervised)} \tag{3.19}$$

$$R \cup N = I \tag{3.20}$$

$$R \cap N = \{\} \tag{3.21}$$

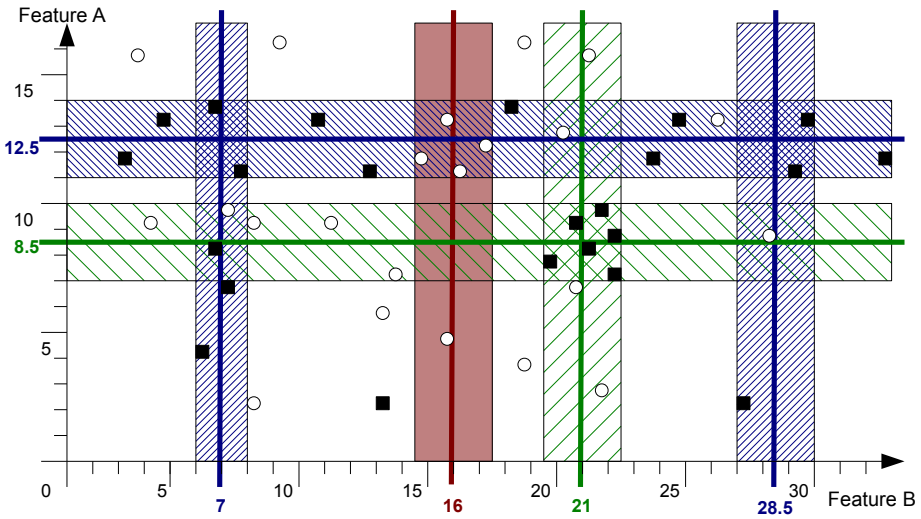


Figure 3.5.: Feature Space Separation
Description of the relevant documents (black squares) by the use of features A and B. Irrelevant documents (white circles) should be excluded. Each box depicts a describable cluster.

3.7.5. Example

A schematic visualization of the concept is depicted in fig. 3.5. It shows a 2 dimensional space, where each dimension represents the similarity for a single feature. The simplified example features A and B are extracted from each image and are represented by a single natural number. The similarity between two feature vectors is a direct mapping from the distance: $s_{xy} = |f(i_x) - f(i_y)|$. The features A and B are assumed to be largely independent, i.e. they are not (closely) related to each other. An example for sufficiently independent features are a "colour histogram" and the "contour" of an object which have no direct relationship among each other.

Relevant images are represented by black squares, irrelevant images by white circles. The aim of the learning algorithm is to find a query, which contains as few parameters as possible covering as many relevant images as possible. At the same time, the amount of irrelevant images should be reduced to a minimum.

In the given example, the relevant images are forming clusters along certain values of each feature. This is an expected and necessary characteristic, which is required to learn a concept in the proposed way. It allows to describe a high-level concept with a

set of low-level features. The resulting optimized query could look like this:

$$
\begin{aligned}
&(f_A : [12.5]@1.5\ NOT\ f_B : [16]@1.5)\ OR\\
&f_B : [7]@1\ OR\\
&f_B : [28.5]@1.5\ OR\\
&(f_A : [8.5]@1.5\ AND\ f_B : [21]@1.5)
\end{aligned}
$$

Every term of this query is used to define a bounded area, where relevant images are located. The second and third term are the easiest ones. In both cases, they cover a reasonable amount of hits (5 respectively 3) and only a single false positive each. The fourth term exploits a useful coincidence, as the images satisfy both features at the same time very well. In that case, the target area can be reduced effectively by intersecting both features into a single result set. Term one illustrates a common problem when using low-level features. The term $f_A : [12.5]@1.5$ describes a large number of relevant images. Unfortunately, about one third of the results are in fact irrelevant. A considerable part of the unwanted result can be described by feature B. Subtracting the hits which satisfy the term $f_B : [16]@1.5$ removes two thirds of the false positives, boosting the precision of the remaining cluster.

The final result set based on the assembled query contains 22 relevant images and a single miss. In addition, this query generated only 4 false positives. This can only be achieved by a clever combination of both features instead of using a single one. Finding a suitable query is a typical optimization problem.

3.7.6. Interpretation as Decision Tree

Building a categorization query can be interpreted as the process of constructing a decision tree. A single binary tree represents the likelihood that an image belongs to the learned concept.

3.7.6.1. Nodes

Each node ν in the tree contains a set of retrieved images $\iota \subseteq I$, which are either a representative of the concept or not. During training, the precision $\pi(\nu)$ and recall $\rho(\nu)$ of each node can be calculated, as the relevant training images T_ν are known.

$$\Pi(\nu) = \frac{|T \cap \iota|}{|\iota|} \tag{3.22}$$

$$P(\nu) = \frac{|T \cap \iota|}{|T|} \tag{3.23}$$

$$\pi(\nu) = \frac{|T_\nu \cap \iota|}{|\iota|} \tag{3.24}$$

$$\rho(\nu) = \frac{|T_\nu \cap \iota|}{|T_\nu|} \tag{3.25}$$

$$\lambda = F_\beta \tag{3.26}$$

$\pi(\nu)$ (eq. (3.24)) is a measure for the categorization strength of a single node. If $\pi(\nu) = 1$, the node only contains relevant images and if $\pi(\nu) = 0$, the node does only contain irrelevant ones. In both cases, the nodes can be labelled as positive ν^+ or negative ν^- example. In reality, the precision is likely to be somewhere between these two extremes, which indicates an uncertainty $\nu^?$ about the categorization. The precision is interpreted as the equivalent to the local information gain in the ID3 algorithm by Quinlan [101]. The recall $\rho(\nu)$ (eq. (3.25)) is less important for each single node, as it is likely, that other nodes are containing some of the relevant images. A derived quality λ (eq. (3.26)), e.g. the F-Measure by van Rijsbergen [136] can be calculated for each single node. If it is based on the based on the precision and recall for the original training set T, λ is a measure for the global classification strength of a single node.

3.7.6.2. Node Splitting

Each node which cannot definitely be categorized as positive or negative example needs to be examined in more detail. A split criterion has to be found to divide the node into two more distinctive sub nodes. This criterion can be defined by a CBIR-query q over the still uncategorized ι_{parent} images. The query generates a result set $r(q)$ and the remaining data $r(q)^{-1} = (\iota_{parent}) \setminus r(q)$. These two sets are representing the content of the two child nodes. Assuming a strong query q, the set $r(q)$ should have a higher precision and is stored in the "left" sub node. The set $r(q)^{-1}$ should have a lower precision than the original set and is stored in the "right" sub node. The aim is to push the precision as close as possible to either 1 or 0 in order to label the nodes accordingly. The "left" nodes should move towards 1 and the "right" ones towards 0. The splitting can be recursively performed until one of the following situations occurs:

1. The node contains no more images
2. The precision π of the node either reached the upper or lower boundary
3. Generating new splitting queries is not possible

These situations are "hard boundaries", where further splitting is not possible or required anymore. For practical use, working against these boundaries may be unrealistic and cause an extremely long or even infinite processing time. For that reason, a set of arbitrary thresholds should be defined, e.g.:

1. Maximum query size
2. Maximum tree depth
3. Minimum information gain on a split
4. Upper threshold of π to flag a node as positive: ν^+
5. Lower threshold of π to flag a node as negative: ν^-
6. Lower threshold of similarity s to create split query q

After termination, the resulting binary tree should have all uncertainties resolved in the leaf nodes. The "left" leaves are considered to contain positive examples and the "right" leaves negative examples for the learned concept. A possible decision tree for the example (section 3.7.5) is shown in fig. 3.6.

3.7.6.3. Root Node

The root node ν_{root} contains all images used in the learning process, both relevant T and irrelevant N ones. For this node, the precision can be calculated as follows.

$$\pi(\nu_{root}) = \frac{|T \cap I|}{|I|} \tag{3.27}$$

$$\rho(\nu_{root}) = \frac{|T \cap I|}{|T|} = 1 \tag{3.28}$$

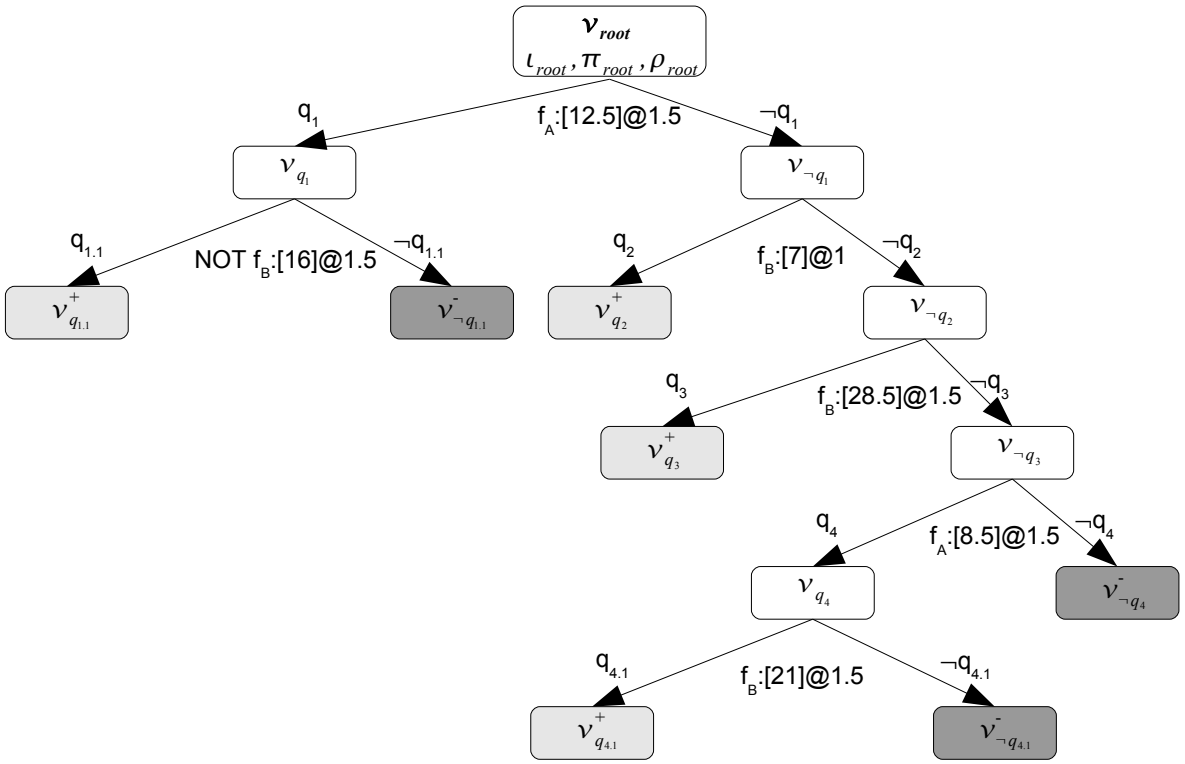


Figure 3.6.: Decision Tree

3.7.6.4. Null Query

The learning algorithm starts with an empty query q_{null}. By definition, this query is undefined and always returns an empty result set $r(q_{null}) = \{\}$. Splitting by q_{null} generates an empty child node $\nu_{q_{null}}$ and another node $\nu_{\neg q_{null}}$ containing the same images as its parent. It follows $\nu = \nu_{\neg q_{null}}$. The root node can be interpreted as the "right" child of a parent node. The null query is also an example for an infinite recursion that can occur during learning.

3.7.7. Learning Algorithm

Quinlan [101] published an algorithm for building decision trees from a given training set. These trees are based on a probabilistic model and usually have problems with inaccurate or unclear definitions. A fuzzy approach by Yuan and Shaw [146] tries to reduce this issue. Instead of using hard boundaries for separating nodes, a less strict separation is used.

In comparison, the learning algorithm 2 suggested in this thesis determines boundaries to split nodes into relevant and irrelevant data. These boundaries are represented by a

Algorithm 2 $findClassificationQuery(T, N)$

```
1: all A ← T
2: result ← {}
3: repeat
4:    query q, λ, TruePositives TP, FalsePositives FP ← findBestQuery(T, N)
5:    if q = {} or |TP| < minTruePositives then
6:       return result
7:    end if
8:    q, λ, TP, FP ← addMustClauses(q, λ, TP)
9:    q, λ, TP, FP ← addMustNotClauses(q, λ, A, FP)
10:   result ← result ∪ q
11:   T ← T \ TP
12: until (|T| < minTermSize)OR(|result| > maxQuerySize)
13: return result
```

single threshold which is set at a point, where the system assumes a reasonable trade-off to get a highly relevant image set after the split. As this threshold is used to cut off less relevant results in each single retrieval process, this boundary is not necessarily fixed. It can easily be moved towards creating either a smaller or larger result set, if required.

The algorithm tries to find a good, but not necessarily optimal query for a given concept. It is constructed to generate results in an acceptable amount of time that allows for realistic testing (i.e. ideally run times not longer than a day). The main aim of this algorithm is to examine the potential of the implemented feature vectors and the query language for categorization tasks.

A concept is fed to the algorithm by providing a representative training set T and the negative set N. The resulting query should return a result set with a very high precision and recall for the target concept.

The training set T is copied to the local set of all relevant images A. Based on the provided data, the algorithm repeatedly tries to determine the best query for T and N. If the sub algorithm 3 is returning a query q and the true positives TP size is above the threshold $minTruePositives$, it is attempted to refine it further. The corresponding quality *quality* and the true and false positives TP and FP are used to search for *MUST* and *MUST_NOT* clauses that may extend q. The resulting q is then added as a *SHOULD* clause to the final *result* and the successfully retrieved TP is removed from T. This loop ends if the size of T reaches $minTermSize$ or the result exceeds a previously set $maxQuerySize$. The conditions introduced in this algorithm are necessary to avoid over fitting.

The algorithm 3 attempts to find the best atomic query q for the given T and N. This is done by generating all possible stub queries q for each training image $t \in T$ and each feature vector $f \in F$. The sorted retrieval results S are then analyzed to find the best similarity threshold. Starting with a size of 2 and ending with $min(|T|, |S|)$ (precision is expected to drop at bigger sizes), the result set size with the highest quality (e.g. F-Measure) is determined. The corresponding similarity value s for the lowest rank is then set as the threshold of the optimized query q. Finally, the best query is returned.

Algorithm 3 $findBestQuery(T, N)$

```
1: q_result ← {}, λ_result ← 0.0, TP_result ← {}, FP_result ← {}
2: for all t ∈ T, f ∈ F do
3:    q ← f(t)
4:    S ← getResults(q, T ∪ N)
5:    s, λ, TP, FP ← calculateThreshold(S, T, N)
6:    q.threshold = s
7:    if λ > λ_result then
8:       q_result, λ_result, TP_result, FP_result ← q, λ, TP, FP
9:    end if
10: end for
11: return q_result, λ_result, TP_result, FP_result
```

3.7.8. Complexity

The proposed algorithm contains 2 main loops and at least one time consuming retrieval process. The execution of the *repeat-until* loop can be controlled by many factors. In essence, every iteration adds a clause to the query and further refines the result quality. For practical purposes, the amount of clauses should be limited to counter over fitting. From a complexity point of view, this loop is independent of the size of the image repository I. Thus, the runtime of this loop is considered to be constant ($O(1)$).

The crucial runtime is within the *for* loop. It iterates through all permutations of training images and features, raising the complexity with both the amount of training images $|T|$ and features $|F|$ involved. In the worst case, the complexity of this loop is $O(|T| * |F|)$. Realistically, the amount of features used is limited and the training set should be small. The most crucial factor in the training algorithm is the *getResults()* function, which performs the CBIR queries. If this function does not use an index, the execution time is linear, as every single image in the repository needs to be compared to the query. The index-less complexity would be $O(|I|)$, but in some cases, the index

could guarantee a complexity of $O(1)$.

The optimization functions *addMustClauses()* and *addMustNotClauses()* both contain a restricted version of the learning algorithm. They are only called for already found base clauses. Thus, in the worst case, their complexity is the same, but usually a small subset of images needs to be checked. By adding sophisticated caching mechanisms and reuse of previous result sets, the execution time could be reduced significantly.

3.7.9. Query Descriptors

The proposed learning algorithm is one possible way to generate a feature-based query for each category to be learned. This query can be considered to be a low-level "descriptor" for a category. Using it in combination with an unknown image, a fuzzy probability of belonging to a certain category could be calculated. This value could be determined by finding the single best describing clause in the query descriptor or by applying the whole query descriptor to the feature vectors of the unknown image. Query descriptors are defined as:

$$\omega = (\mathrm{q}, \mathrm{category}) \tag{3.29}$$

$$\Omega = \{\omega_1, \ldots\} \tag{3.30}$$

where ω is a tuple of the query q and the category being represented. Ω is the set of all query descriptor tuples.

Classifying unknown images with a set of query descriptors encounters similar problems as in biometric systems with multiple sensors. Jain et al. [59] describe several types of information fusion, split into "pre-classification" and "post-classification".

3.7.9.1. Pre-Classification Fusion

The first stage of fusion occurs when an unknown image is fed to a single query descriptor. This is done by a "feature level fusion" [59]. Instead of creating a simple "weighted summation" or "concatenation" of the features, the calculations in a query descriptor are more complex. Each descriptor contains similarity thresholds, weights and Boolean rules to combine the results from the basic features. The resulting probability ϕ is a value in the range $[0.0, 1.0]$.

A drawback of this approach would be the use of multiple feature vectors from dif-

ferent domains that are insufficiently fine-tuned towards each other with respect to the similarity measures. While certain feature vectors may rank many images with values above 0.99, another one may only assign a value of 0.75 to highly similar images. This problem is addressed in section 6.2.

Further, the thresholds of the Boolean clauses in the query need to be considered. Their initial purpose is to suppress all results below that threshold from appearing in the result set of a search. The threshold of each positive clause is chosen in a way to be a reasonable trade-off between a high precision and recall in the remaining results. Thus, many relevant images may still lie below this threshold which is especially true for unknown objects from the same category. A classificator should still be able to recognize these relevant images. The retrieval algorithm simply pulls the similarity value of images below the threshold to 0.0, effectively removing them from the result set. The classification is less radical and min-max normalizes the calculated similarity according to the threshold θ.

$$\phi(\omega, i) = \text{normalize}\,(s_{\text{q}}(i), \theta) \tag{3.31}$$

$$\text{normalize}\,(x, \theta) = \begin{cases} \frac{1}{1-\theta}(x-\theta) & \text{if } x \geq \theta \\ \frac{0.5}{\theta}(x-\theta) & \text{if } x < \theta \end{cases} \tag{3.32}$$

where $s_{\text{q}}(i)$ is the calculated similarity between query and image i, $\phi(\omega, i)$ is the probability of i to be a member of the category of the descriptor q and θ is the top level similarity threshold within q. The normalized probability is set to be 0.5 at θ by using linear interpolation. This normalization is performed recursively for each Boolean clause in the original query. As a result, the probability value is a direct indication as to how well an unknown item matches the descriptor. If $\phi \geq 0.5$, the image would appear in the result set of the query and if $\phi < 0.5$ it would be removed. The set of all probabilities Φ is defined as:

$$\Phi(i) = \{\phi(\omega, i) |\, \omega \in \Omega\} \tag{3.33}$$

where $\phi(\omega, i)$ is the probability of image i to be described by the query in ω,.

3.7.9.2. Post-Classification Fusion

At the second stage, multiple probabilities ϕ_{q} need to be considered and the system uses the information to make a decision. In the pure classification use-case, the output is a single category to describe the previously unknown object. A sophisticated fusion of probabilities is not required and a "dynamic classifier selection" also known as "winner-takes-all" [142] is applied. The major disadvantage of this approach is the loss of accuracy by ignoring other categories which also may have high probability values:

$$\mathrm{category}(i) = \left\{ \mathrm{category} \,|(\mathrm{q}, \mathrm{category}) = \omega \wedge \phi(\omega, i) \arg\max_{\phi} \left(\Phi(\omega, i)\right) \right\} \tag{3.34}$$

where $\mathrm{category}(i)$ is the set of categories that are assigned to the image i. In this case, it contains the category with the highest probability.

In retrieval related use-cases, it is desirable to tag unknown images automatically with a set of possible labels. Various systems automatically assign label sets to unknown images, assuming that most of them are correct, allowing the user to find those images by label while accepting a couple of false positives in the result [61, 126].

Chapter 4.

Design

In this chapter, a CBIR design based on a previous masters thesis of the author [90] (see section 2.2) is presented. The existing system already allows for a retrieval by a query string specified in a recent journal article [96]. The system has been extended to support alternative user interfaces as well as a testing module used in the case studies in chapter 6.

4.1. Retrieval Framework Design

Most retrieval systems follow a basic work flow cycle (see section 3.1). A successful retrieval is performed as follows: users submit a query to the engine to trigger a search and receive a set of results. Those results may be satisfying and the user finds the required information. If not, the user may navigate through the results and eventually refines the previous query to trigger a new search.

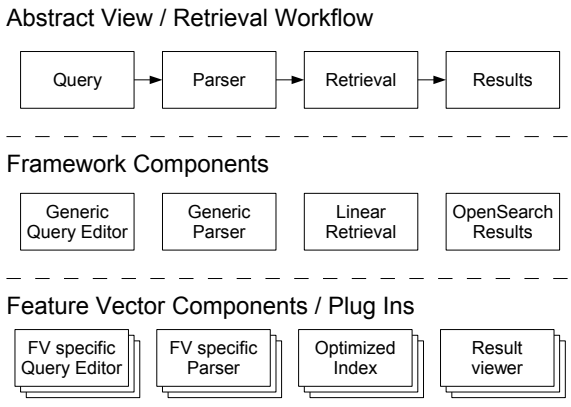


Figure 4.1.: Layers in the Retrieval Process [96]

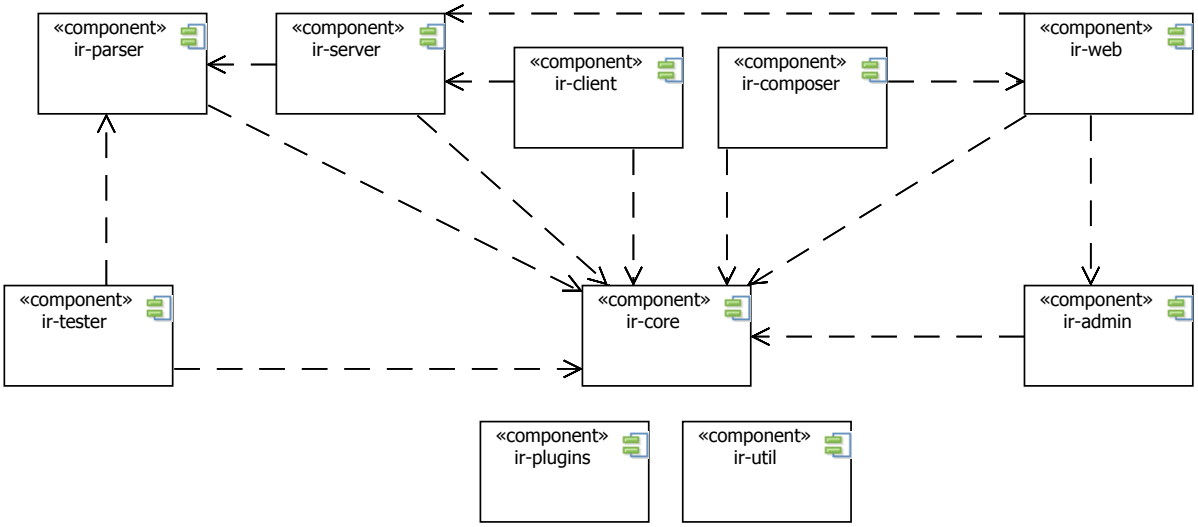


Figure 4.2.: Main Components

Figure 4.1 illustrates the internal search steps performed by the retrieval engine. Three different abstraction levels are employed [96].

At a very high abstraction level, a simple retrieval system receives a query from the user, parses it somehow to understand the meaning, gathers the most relevant documents and finally returns them. This work flow is very common and can be offered by a generic framework, which simply offers all the basic functionality required. These framework components do not have to be specialized. They only need to understand standardized input and generate standardized output. All the details and optimizing are meant to be implemented in exchangeable plug-ins on the use-case dependent implementation level.

4.1.1. Main Components

The main components of the system and their dependencies are presented in fig. 4.2. This diagram is a simplification of the true dependencies to ensure readability. Every single component makes use of “util”. Several system components developed for the preceding master thesis have been extended within this thesis: “core”, “admin”, “client”, “util”, “plugins”, “web” and “server”. These have mainly been extended by an optional indexing structure for the plugins, XML support and additional interfaces. Completely new are the components “parser”, “tester” and “composer”.

The “core” component contains all CBIR data structures. It also defines the generic interfaces for the “Framework Components” in fig. 4.1.. The “util” is a loose collection of several useful methods that are required in several other components. The “plugins”

are a collection of optional feature vector implementations. They are essential to make the CBIR work, but the choice of these plug-ins is up to the user.

The other components provide the actual realizations for work flow and user interfaces. "server" and "parser" are purely to provide the work flow. The "server" encapsulates the linear retrieval work flow of fig. 4.1. It accepts queries, forwards it to the "parser", performs the retrieval and returns the results.

The "admin" component encapsulates the functionality to manage the image repository. The main tasks are adding new images and extracting the feature vectors. This information is written into the repository for later use by the retrieval software.

The retrieval components usually require read-only access to the repository (without relevance feedback). Two retrieval clients have been developed by the author of this thesis prior to this PhD work. A fat client solution is contained in the "client" package, containing a full graphical user interface to access the "server". A thin client is provided by the "web" package, allowing to use the "server" without the need of installing special software on the client machine. The UI of this package is supposed to be minimalistic, containing a user interface for directly submitting query strings and displaying results. On top of this service, a graphical query building software — the "composer" — is provided.

A special component is the "tester". It contains a collection of time-consuming algorithms for analyzing the repository and also for learning the query descriptors.

4.1.2. Speed & Quality

The earlier system developed by the author had been optimized for a high accuracy rather than a high processing speed [90]. Each retrieval request is processed linearly by comparing the query feature vector to each stored feature vector. To provide a better scalability, the system is extended to support arbitrary indexes. Depending on the type of feature vector, an extremely feature specific index is required (section 2.3.4). However, a high accuracy is more important to optimize the feature normalization and learning methodology than a short response time. No user is required to be operating the system after starting a test, thus the algorithms are able to run automatically during the night.

4.2. Query Language

The query language in the proposed system (section 2.2.1) [96] is based on the Lucene Query Syntax [6]. It is intentionally chosen to provide beginners with a simple and familiar syntax.

4.2.1. Wildcards and Ranges

Wildcards and ranges can be used to express uncertainty or to allow the search engine to be less strict during retrieval. The meaning of those concepts depends on the described feature. Some features may well benefit, but for others they may not be required.

In text retrieval, wildcards stand for letters in a word that don't have to match an explicit query. In the example case of a RGB mean value, a wildcard can express, that a certain colour channel does not need to be considered. For spatial features it can be useful to define regions of interest as well as regions of non-interest.

Ranges are an intermediate concept between concrete queries and wildcards. They are used to specify a certain space where parameters can be matched. Searching for images with a creation time stamp is only feasible, if a range can be specified. It is very unlikely, that the searcher knows the exact time, especially when it is extremely accurate (i.e. milliseconds). In such a case, usually a time span is provided (e.g. "between 03/06/08 and 10/06/08" or "within the last week"). Analogous, image features such as the trivial RGB mean could specify a tolerance range for each colour channel.

Unfortunately, these definitely useful concepts cannot be fully generalized. At this point, the plug-in developer needs to decide how to address them. Taking the RGB means of an image, the user could specify an array like *"[36, 255, *]"*. In this case the results should contain some red and dominant green. The rate of blue does not matter at all. Putting some more effort into the feature abstraction, a more convenient query like *"some red and very much green"*is also possible. This lies in the responsibility of the plug-in developer.

4.2.2. Parse Trees

Based on the grammar, the parser generates a hierarchy of sub queries wrapped up in a single root query object. By traversing the tree, the sub results can be merged accordingly. This section shows the decomposition on a relatively complicated query. The images used in this example are part of the Caltech-101 collection [41].

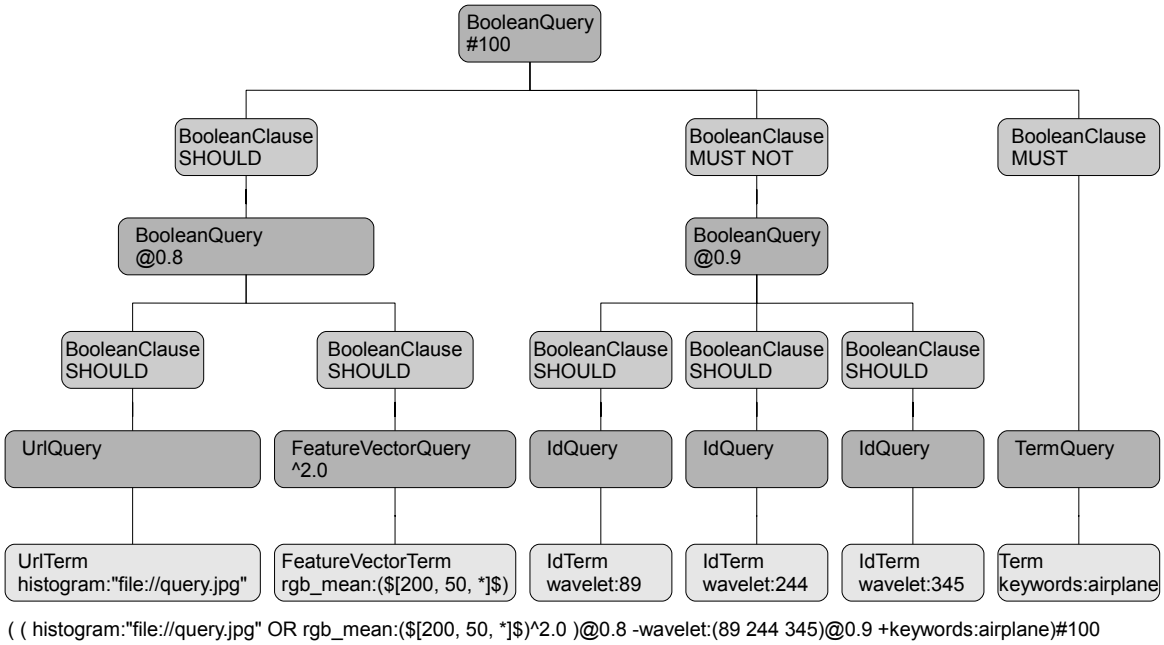


Figure 4.3.: Parse Tree of a complex Query

```
(
 (
  histogram:"file://query.jpg" OR
  rgb_mean:($[200, 50, *]$)^2.0
 )@0.8
 -wavelet:(89 244 345)@0.9
 +keywords:airplane
)#100
```

Verbally, this query can be read as follows:

> "Find images, that have a similar histogram as the sample image *query.jpg* OR have a mean colour close to *200 red* and *50 green*. The blue channel can be anything. Rank the mean colour twice as high as normal. Both sub results should have at least a similarity value of 0.8. Please remove any result, that has a minimum wavelet similarity of 0.9 to the images 89, 244 and 345. Every result must be annotated with the keyword *airplane*. Find not more than 100 results in total."

After parsing, the query string is converted into a parse tree that contains all of the relevant concepts (fig. 4.3). The root node is represented by a *Query*, which is the single

data object that is processed by the retrieval core. Each leaf is a *Term*, representing a partial search, which generates a sub result. The tree structure in between represents the rules how to merge the sub results into a final one.

The search engine then traverses the tree and generates the answer to this particular request. At this point, it is advisable to integrate a query optimizer to reduce the response time. In the current prototype, some straightforward query optimizing already takes place.

First, the *MUST* clauses are processed, then the *SHOULD* and finally the *MUST_NOT* clauses. This allows for an early reduction of the search space, which is especially of importance, if no index or only a slow index is available for certain features. Depending on the availability of indexes, the terms with the shortest processing time should be preferred. The optimization strategy should always be aimed at an early reduction of search space as well as preferring the use of fast indexes. The strategy applied in this case uses a strict definition of *MUST* and *MUST_NOT*. If an image is not part of all the *MUST* clauses or part of a *MUST NOT* clause, it is removed from the final result. This approach is considered to be a useful trade-off between a perfect fuzzy algebra and speed optimizations.

In this case, the first term to be processed is "keywords:airplane". This triggers a keyword search, which is backed by a fast and efficient index, resulting in a list of matching images. As the parent *BooleanClause* is flagged as *MUST*, the final results of the query can only be amongst those sub results. Assuming, that only about 1% of the repository is related to the keyword "airplane", every subsequent linear search time can also be reduced to only 1% of the otherwise total scan time.

The second branch to be processed is the *SHOULD* clause on the left, that is split into a nested boolean query.

One leaf contains a *UrlTerm*, pointing at an external query image and requesting a comparison based on its histogram. To process this part, the engine reads the image from the URL and extracts the histogram automatically. This search only needs to compare the query histogram with the stored histograms from the previous sub result.

The other leaf contains a *FeatureVectorTerm*. The string embedded between the "($" "$)" brackets is parsed by the *rgb_mean* plug-in. In this case, the string stands for the three mean colour values *red* (200), *green* (50) and *blue* ("don't care" wildcard) of an image. Again, the search space is drastically reduced by the first sub result.

After both terms have been processed, the sub results are merged into a single one. Their combined similarity must be at least *0.8*, otherwise the image is removed from

the result set. There is no "best" rule to merge the sub results. Within this thesis, the algorithm of merging described in section 3.6.2 is used.

In this case, the *rgb_mean* branch has a weight of *2.0* and thus gains a higher importance in the merged result.

The last main branch is flagged as *MUST_NOT* and requires a minimum combined similarity of *0.9*. All of the three clauses contain a plain *IdQuery* with an *IdTerm*. They require a retrieval on the wavelet feature and use sample images from the repository by stating the image id directly. Again, the search space is already limited, not only by the *MUST* branch, but also by the *SHOULD* branch. It is only necessary to check the images contained in the previously retrieved sub result. The sub results of the middle branch are merged accordingly and cropped at a minimum similarity of 0.9.

To generate the final answer, the *MUST_NOT* results are removed from the temporary sub result. The last step required is to cut the sorted list after the 100 best hits.

4.2.3. Browsing

This section analyses, which parts of a retrieval session can be realised with the chosen query language described in section 4.2. For reference, the model by Garber and Grunes [44] (fig. 2.2) is used.

To have some choice, some global features are assumed to be available. The feature related strings are bracketed within *($* and *$)*.

Generate criteria and weightings This can be done in several ways, depending on the feature extensions available.

Find red images:

```
fv_mean:($[255,0,0]$)
```

Find either an orange image or a fruit with double weight on the colours:

```
fv_mean:($orange$)^2 keywords:"fruit"
```

Enter picture and similarity criteria Find similar images to 123 based on wavelets (QBE):

```
fv_wavelet:123
```

Find similar images to the image "http://example.net/query.jpg" based on its histogram (QBE):

```
fv_histogram:"http://example.net/query.jpg"
```

Retrieve set of images This is the normal behaviour of each retrieval system. A set of results is generated based on the query.

View images and information about the images Viewing details about images is part of the user interface. Nevertheless, the meta information stored in the index should be readable.

Put on or remove from light table, etc. Again, this is part of the user interface, not of the query processing.

Add criterion This can be done by adding a boolean term to the existing query.

Find images that are orange AND are tagged with the keyword "fruit".

```
fv_mean:($orange$) AND keywords:"fruit"
```

Add restriction Restrictions can be added by specifying a NOT term.

Find images that are wavelet-similar to image 123 and remove images that have more than a 0.9 histogram similarity to image 987.

```
fv_wavelet:123 NOT fv_histogram:987@0.9
```

Change criterion Changing a criterion is equivalent with changing the contents of a term. This could be the field/feature name or the data to be found.

Re-Define search Resetting the query and starting from scratch is no problem, as long as each search request is a single transaction.

Pull target image This final step is again in the responsibility of the client software. If the results are returned with the URL of the images, this is easy to implement.

Most of the browsing aspects by Garber and Grunes [44] can be represented by the chosen query language. Some functionality also requires support by the browser application itself, such as marking and remembering possible hits.

4.3. Learning Algorithm

The learning algorithm described in section 3.7.7 attempts to find the optimal result within the given parameters, e.g. $minTermSize$, $maxQuerySize$. To keep the complexity within reasonable boundaries and to prevent over fitting, these values are set manually.

The weighting factor β between precision and recall to measure the quality of each tested query (eqs. (3.4) and (3.26)) is set to $\beta = 0.2$. This effectively shifts the threshold for each clause to a high value, resulting in small result sets with a high precision .

Concerning top-level SHOULD clauses, three parameters are set. In order to be able to ignore outliers, the amount of true positives found by a SHOULD clause must be $|r(c)| \geq 5$ to be included in the query. Equally, the training set must have at least a size of $|T| \geq 5$. A break condition in the main loop is the maximum amount of iterations generating positive SHOULD clauses. It is restricted to $|C| \leq 20$ to keep the query descriptors at a reasonable size and also ensuring reducing the maximum processing time of algorithm 2.

Similar considerations are valid for the additional MUST_NOT clauses. The amount of clauses added to each top level clause must be $|C^-| \leq 5$ to prevent a high amount of negative clauses. The training set must have a size of $|r(c)| \geq 2$. This value has been chosen, as tests indicated that often only a small group of related false positives appeared within the results. Similarly, each clause must remove at least $|r(c^-)| \geq 2$ false hits, i.e. every negative clause must remove at least two false positives from the intermediate result.

The last set of rules has been set for the MUST_HAVE clauses. Constructing an intersected pair of results should always happen by using an unrelated feature vectors. Using two independent feature vectors would restrict the search space in both feature spaces and make the resulting intersection very robust (sections 3.7.2 and 3.7.5). As a very hard condition, not a single true positive must be lost by using a second clause. This prevents the algorithm to accidentally cutting out relevant images. Further, the amount of MUST_HAVE clauses in the final query indicate, how well two independent feature vectors are able to capture the same concept.

4.3.1. Multi Threaded Processing

Most tests can be parallelized. Every test with a high processing load is split into smaller work units. Dependent on the available processor cores and memory available,

a semaphore (typically set to $2 * cores$) controls the amount of parallel threads. This is especially necessary when having a large number of threads or resource intensive ones. The default pattern works as follows:

1. Initialize system (e.g. CBIR service)
2. Assemble list of tasks to be done
3. Iterate through tasks
    a) Create runnable object ("work unit")
    b) Initialize work unit
    c) Wait for available semaphore
    d) Start work unit as thread
4. Wait for all threads to finish
5. Clean up system (e.g. database connections)

Chapter 5.

Implementation

Most of the system is implemented in Java 1.6. The web user interfaces are based on Servlets to generate HTML containing some JavaScript.

During the case studies, several JUnit tests were generated for quick independent test runs. For the flexible analysis of many test results, a couple of Matlab scripts are also used.

Below a list of the relevant sub projects. Each project is implementing a component described in section 4.1. Several existing sub projects already implemented within the author's master thesis [90] are used as foundation. They are extended to meet the additional requirements of the new methodology:

ir-core classes and interfaces to access image data and feature vectors

ir-admin tools to add new images to the repository

ir-client Java Swing based client for query composing and retrieval

ir-web Apache Tomcat based client for retrieval

ir-echo2 Echo2 Web Framework based client for query composing and retrieval ("composer")

ir-parser Apache Lucene based query parser

ir-server retrieval and ranking algorithms

ir-util collection of various utility classes

ir-tester tools for automated testing and learning algorithms

ir-plugins set of various feature vector plug-ins

ir-plugin-keyword keyword feature, based on the file path and name

ir-plugin-mean RGB mean value

ir-plugin-stochastic2 12 stochastic moments of image histograms [4]

ir-plugin-stochsticquad like *stochastic2*, image split into quad tree [92]

ir-plugin-wavelet list of the highest Haar-wavelet coefficients [58]

5.1. Query Language

The system uses a modified version of the original Lucene Query Parser [96]. The parser analyses an input string and converts it into the corresponding *Query* object. Dependent on the terms given, the *Query* is composed of different clauses. This could be a single one, an array of Boolean clauses or may be even nested. The language allows queries similar to those used in traditional search engines and the parser is generated by JavaCC.

The string representation of a query can be manipulated directly by users. This query string can be edited in every basic text editor without the need for any extended user interface. It also allows experienced users to access every aspect of the search engine directly.

As the query language is based on the Lucene tool kit, it always has an object representation of the whole query. This query object could also be created by a suitable front end. Such a tool eliminates parsing errors, because the query structure would always be bound to the components.

Below, two alternative representations for the query language are presented. As an increasing amount of XML based data processing is expected — such as MPEG-7 [75] and MRML [81] – it seems beneficial to represent queries in an Extensible Markup Language (XML) structure also. Further, the use of QBE, QBF and especially QBS suggest the use of a more visual query representation. Rather than being limited to a textual string, a visual query containing actual pictures could enhance the user's understanding of a query.

5.1.1. XML

The parse tree containing a query (section 4.2.2) can be directly mapped to an XML hierarchy. Plug-ins could also specify their feature conversion into XML and back. This allows for consistent XML files and simplifies the use of the MPEG-7. Below, the XML structure matching the example parse tree (section 4.2.2) is shown:

```
<boolean-query max-count="100">

  <boolean-clause occur="SHOULD">
    <boolean-query threshold="0.8">

      <boolean-clause occur="SHOULD">
        <url-query>
          <url-term>
            <field>histogram</field>
            <url>file://query.jpg</url>
          </url-term>
        </url-query>
      </boolean-clause>

      <boolean-clause occur="SHOULD">
        <feature-vector-query boost="2.0">
          <feature-vector-term>
            <field>rgb_mean</field>
            <string-data>
              [200, 50, *]
            </string-data>
            <data>
              <red>200</red>
              <green>50</green>
              <blue>*</blue>
            </data>
          </feature-vector-term>
        </feature-vector-query>
      </boolean-clause>

    </boolean-query>
  </boolean-clause>

  <boolean-clause occur="MUST_NOT">
```

```
  <boolean-query threshold="0.9">

    <boolean-clause occur="SHOULD">
      <id-query>
        <id-term>
          <field>wavelet</field>
          <id>3960</id>
        </id-term>
      </id-query>
    </boolean-clause>

    <boolean-clause occur="SHOULD">
      <id-query>
        <id-term>
          <field>wavelet</field>
          <id>3941</id>
        </id-term>
      </id-query>
    </boolean-clause>

    <boolean-clause occur="SHOULD">
      <id-query>
        <id-term>
          <field>wavelet</field>
          <id>3948</id>
        </id-term>
      </id-query>
    </boolean-clause>

  </boolean-query>
</boolean-clause>

<boolean-clause occur="MUST">
  <term-query>
    <term>
```

```
        <field>keywords</field>
        <text>airplane</text>
      </term>
    </term-query>
  </boolean-clause>

</boolean-query>
```

This XML data contains the same information as the example query string. Clearly, this format is much more verbose than the suggested query language. Being probably less readable for humans, its advantage is the standardized format. The XML code does not require a special parser to be processed or validated by any program.

One example of generic and specialized data representation is contained in the XML query above. The *feature-vector-term* for the *rgb_mean* plug-in shows two alternatives. In the generic case, the *string-data* is left untouched. This is the output generated by the main parser. To extract the real meaning of the data string, it needs to be processed by the corresponding plug-in. The resulting *data* tag would then contain each piece of feature data separately.

```
<feature-vector-term>
  <field>rgb_mean</field>
  <string-data>
    [200, 50, *]
  </string-data>
  <data>
    <red>200</red>
    <green>50</green>
    <blue>*</blue>
  </data>
</feature-vector-term>
```

5.1.2. Visual Query

A clearly structured query language like the proposed one can be mapped to a visual representation to guide the user. The resulting graphical user interface helps to assemble queries that are syntactically correct, displays query images and provides a canvas

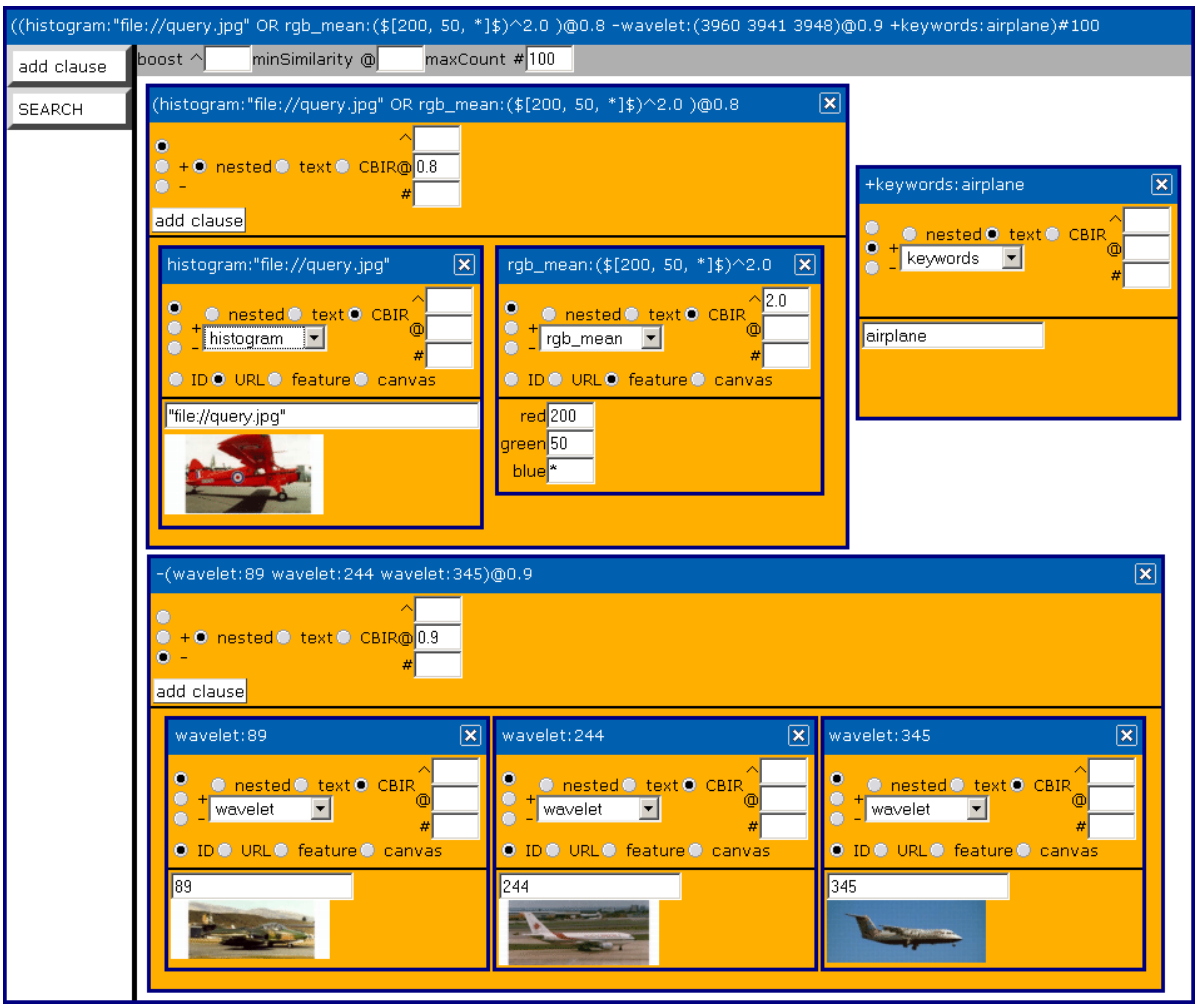


Figure 5.1.: Visual Query Composer

for QBE. Additionally, it may also support the modification of feature plug-in specific parameters.

Figure 5.1 shows the visual query composer of the prototype, where the example query has been assembled from multiple clauses. Every clause of the parse tree is modelled by a window. Each clause window contains several options to choose the occurrence, the added parameters, a field name and the query type. Textual queries usually contain a generic term with one ore multiple keywords.

Queries for CBIR can manage a query URL, a specified feature, the id to an existing image or a canvas to draw a query image. Clause windows specifying a query image can directly display a small preview image to provide feedback what is going to be searched. Windows containing a feature description can either show the data fields directly (e.g. red, green, blue) or a convenient editor (e.g. a colour chooser).

5.2. Algorithms

Below, the Java code for the core changes and additions in the system are listed. For the sake of readability, some parts of the code were omitted. These parts are marked with /* ... */.

5.2.1. Normalization Algorithm

The Listings 5.1 and 5.2 implement the changes required for the feature based similarity normalization described in section 3.4.2.

The method in listing 5.1 is based on the previous ranking code. The additional code `normalizeSimilarity(rawSimilarity, fvNormValues)` is to normalize the calculated similarity from the feature vector according to a previously determined sorted set of normalization values.

Listing 5.1: getRanking

```
private ResultList getRanking(FeatureVector query, int maxSize,
    double threshold, Iterator<DataItem> it) {
  SortedSet<ResultItem> sorter = new TreeSet<ResultItem>();
  /* ... */

  // compare FVs with input and add them sorted to the result
  while (it.hasNext()) {
    // get next stored feature vector
    FeatureVector storedFV = (FeatureVector) it.next();
    if (storedFV != null){
    // calculate ranking
    double rawSim = storedFV.calculateSimilarity(query);

    // normalize similarity by fvNormValues
    double normSim = normalizeSimilarity(rawSim, fvNormValues);

    ResultItem ranked = new ResultItem(normSim, storedFV.getId());

    // add to sorted resultset, if higher or equal threshold
    if (ranked.getSimilarity() >= threshold) {
```

```
        sorter.add(ranked);
      }
      /* ... */

      if (sorter.size() > maxSize) {
        sorter.remove(sorter.last());
      }
      /* ... */
    }
    return new ResultList(sorter);
}
```

Listing 5.2 adds the mandatory boundaries 0.0 and 1.0 to the provided `normValues` and asserts that no other value exceeds them. The range between 0.0 and 1.0 is split into equi-distant segments. The normalized similarity is interpolated from the two nearest points of the normalization values.

Listing 5.2: normalizeSimilarity

```
private double normalizeSimilarity(double rawSimilarity,
    SortedSet<Double> normValues) {
  SortedSet<Double> ss = new TreeSet<Double>(normValues);

  // define bounds
  ss.add(0.0);
  ss.add(1.0);
  assert ss.first() == 0.0;
  assert ss.last() == 1.0;

  double segmentsize = 1.0/(ss.size()-1);

  // find nearest two values
  SortedSet<Double> smaller = ss.headSet(rawSimilarity);
  SortedSet<Double> greaterOrEqual = ss.tailSet(rawSimilarity);
  double x1 = 0.0;
  if (!smaller.isEmpty())
    x1 = smaller.last();
```

```
  double x2 = 1.0;
  if (!greaterOrEqual.isEmpty())
    x2 = greaterOrEqual.first();

  // determine fractions between 0.0 and 1.0
  double y1 = Math.max(0.0, smaller.size()-1)* segmentsize;
  double y2 = (smaller.size())* segmentsize;

  // Linear Interpolation
  double result = y1 + (y2-y1)/(x2-x1) * (rawSimilarity-x1);

  return result;
}
```

5.2.2. Learning Algorithm

Listing 5.3 is the implementation of algorithm 2. The matching sequence diagram is shown in fig. 5.2. The second listing 5.4 is an implementation of algorithm 3. Figure 5.3 shows the same method formatted as a sequence diagram.

The method in listing 5.3 attempts to create a Boolean query with a set of *SHOULD* clauses. The method uses training set T and the negative set N to determine the best clause. After each clause, the set of remaining training images is reduced by the images found.

Listing 5.3: findClassificationQuery

```
public Query findClassificationQuery(IdList T, IdList N) {
  BooleanQuery result = new BooleanQuery();
  /* ... */
  // REPEAT
  do{
    QualityParameters bestQP = findbestQP(T, N);
    /* ... */
    Query bestQuery = bestQP.getQuery();
    ResultList bestResults = bestQP.getResultList();
    List<ResultItem> sorted = bestResults.getSortedResults();
```

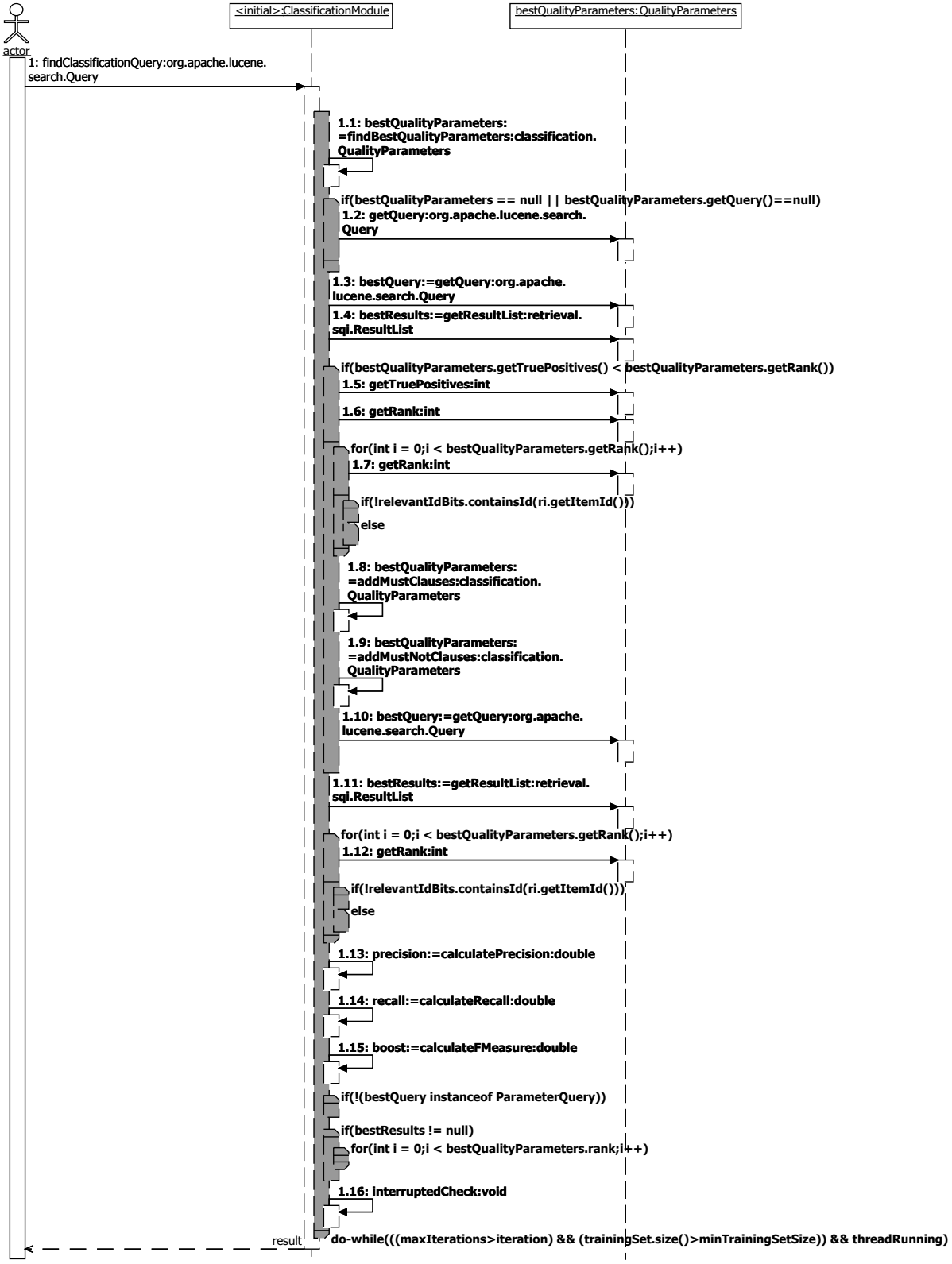


Figure 5.2.: findClassificationQuery - Sequence Diagram

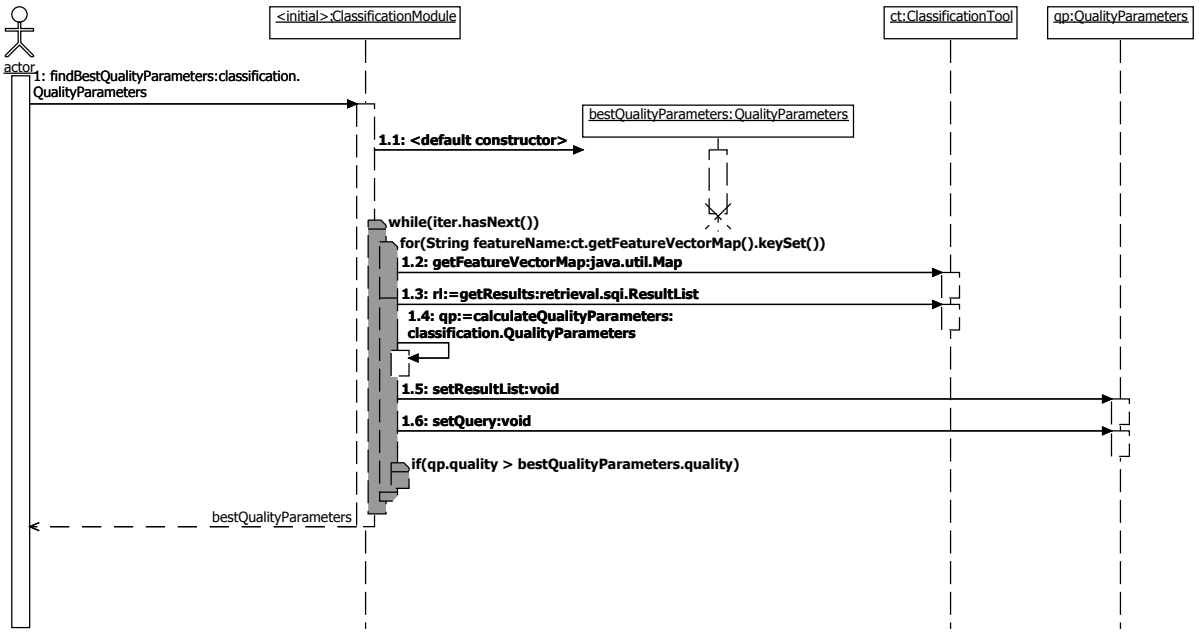


Figure 5.3.: findBestQualityParameters - Sequence Diagram

```
// try to improve query, if too many FP are contained
if (bestQP.getTP() < bestQP.getRank()){
  // TP here: amount of newly added/learned documents
  // all false positives in results
  BitIdList FP = /* ... */;
  // all true positives in results (shrinking training set)
  BitIdList TP = /* ... */;
  //  add MUST/MUST_NOT clauses
  bestQP = addMustNotClauses(bestQP, relevantIds, FP);
  bestQuery = bestQP.getQuery();
}

// determine boost for new clause
...
float boost = (float)calcFMeasure(precision, recall, beta);
// set boost to query
/* ... */
((ParameterQuery)bestQuery).setBoost(boost);
// add found query as OR clause to result query
```

```
    result.add( new BooleanClause( bestQuery,Occur.SHOULD ) );
    // remove found IDs from T
    /* ... */
  }
  // UNTIL no more improvements possible
  while( ((maxIterations>iteration)
      && (T.size()>minTSize)) && threadRunning );

  return result;
}
```

The listing 5.4 attempts to find the query with the best quality with regard to the provided remaining training set T and the negative set N.

Listing 5.4: findbestQP

```
private QualityParameters findbestQP(IdList T,
    IdList N) throws RankerException {
  QualityParameters bestQP = new QualityParameters();
  Iterator<Integer> iter = T.idIterator();
  while(iter.hasNext()){
    int currentId = iter.next();
    for (String featureName : getFeatureVectorMap().keySet()) {
      Query query = new TermQuery(
        new Term(featureName,Integer.toString(currentId))
      );
      BitIdList validIds = new BitIdList(T);
      validIds.addIds(N);

      ResultList rl = ct.getResults(query, validIds);
      List<ResultItem> sorted = rl.getSortedResults();
      // calculate quality (based on F-Measure)
      QualityParameters qp = calculateQP(sorted, T, N);

      ParameterQuery pq = new ParameterQuery(query);
      TermParameters tp = new TermParameters();
      tp.setThreshold((float)qp.threshold);
```

```
        pq.setParameters(tp);
        query = pq;

        qp.setResultList(rl);
        qp.setQuery(query);

        // determine best result
        if (qp.quality > bestQP.quality){
          bestQP = qp;
        }
      }
    }
    return bestQP;
}
```

5.2.3. Multi Threaded Processing

In this section, an example is given for the calculation of query descriptors (fig. 5.4). It follows the generic pattern to assemble work units in a loop and starting a thread for each unit as described in section 4.2.

As soon as the *ClassificationTool* is initialized with working database access, the method *buildDescriptors()* can be triggered externally. In the *FOR* loop, all feature vectors to be used are cached in the memory to provide fast access. The following *WHILE* loop then collects all available IPTC keywords from the textual search index. Each available keyword is used as a category name, i.e. is a work unit to create a query descriptor. The resulting list contains all time consuming tasks to be done.

The final *FOR* loop is now responsible for creating and starting the threads. For each category, a new *ClassificationModule* is created and initialized with an individual category string. Before starting a new thread, the loop locks at the semaphore.

After starting all threads, the system waits for them to finish. A small shut-down thread is used to clean up all open handles as soon as the last work unit terminates.

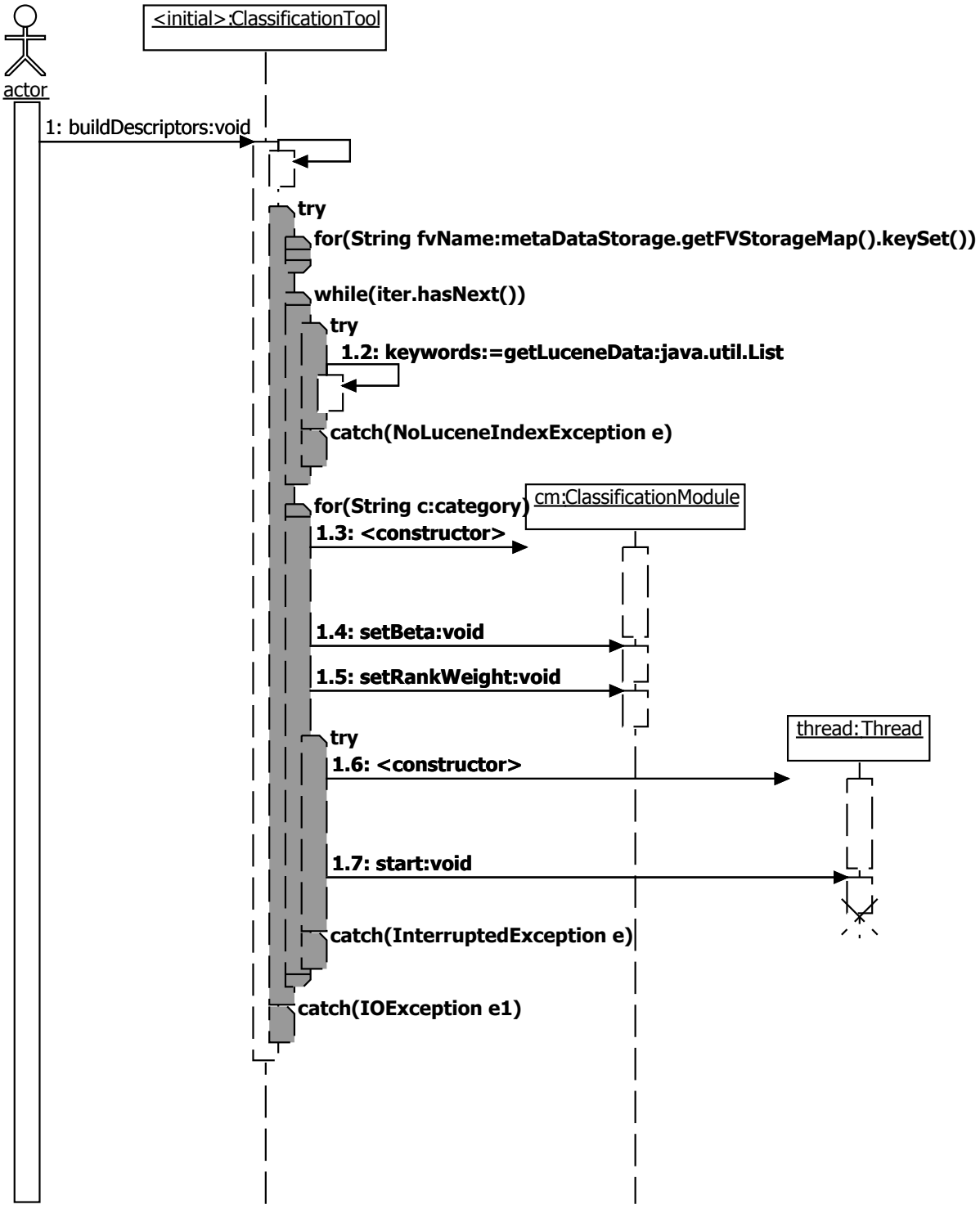


Figure 5.4.: buildDescriptors - Sequence Diagram

Chapter 6.

Case Studies

This chapter discusses several case studies to evaluate the methods introduced in chapter 3. Each case study focuses on a specific issue and the advanced cases build up on previous findings. The first ones are dealing with the low-level feature vectors directly and then the scope widens to the interrelationship of multiple feature vectors until their combination within a single query is used as a mapping rule for higher-level semantics (i.e. categories).

In section 6.1, some individual properties of the feature vectors used are benchmarked in an exemplary way. The methodology is described in section 3.4.1. The aim is to analyze the robustness of the single feature vector according to the image size used in simple QBE queries.

Section 6.2 investigates the feasibility of an empirical feature normalization as proposed in section 3.4.2. The normalization is expected to reduce the differences in similarity calculations among the implemented feature vectors, resulting in comparable similarity profiles. This is especially required for the interpretation of similarities as probabilities and thus a more accurate fusion when using the query descriptors for categorization tasks (see section 3.7.9).

Section 6.3 describes a way to estimate the discriminative power of each single feature vector for all of the given categories within a repository. The derived improvement factor is supposed to indicate which feature vector is expected to be most powerful for use in a query descriptor to be learnt. It basically represents the first iteration of the learning algorithm in section 3.7.7, generating a query only containing a single term.

In section 6.4, a machine learning algorithm based on DTs (section 3.7.2) is applied to construct a set of category related queries. This case study is used to fine-tune the behaviour of the algorithm and to find reasonable parameters to find a reasonable balance between precision and recall and to avoid over fitting.

The descriptors constructed during the learning are used for a simple categorization in section 6.5. Other than performing a full retrieval on the database, the query is only used to categorize a single image into a fixed category. Instead of using a fast fusion approach (section 3.6.2), a more accurate approach is applied (section 3.7.9) to determine the most likely category. The results are not expected to be highly influenced by the similarity characteristics of distinct features, as they have been already normalized (section 3.4.2).

The final case study in section 6.6 wraps up the findings of all previous ones into the semi supervised learning approach described in section 3.7.3. It directly compares the proposed learning approach to an existing one on the same data. A leave-one-object-out cross validation is performed to evaluate, how well an unknown object can be categorized.

6.1. Impact of Query Image Size on Features and Similarity Measures

When performing QBE queries, it is not feasible to always send a high-resolution image to the retrieval service. Thus, the amount of submitted data needs to be reduced. The most efficient way would be to extract the relevant feature vectors on the client-side. This requires to transfer the corresponding executable code to the client. This is difficult to achieve for several reasons. This approach requests additional processing resources from the client and the code must be runnable on the target platform. Further, security concerns may occur.

Another way is to use existing and more generic client-side software to reduce the size of the original image by compression or a lower resolution. The effect of the query image size is evaluated to find practical limitations of downscaling.

6.1.1. Requirements

- default image datasets

6.1.2. Testing

In this study, the Caltech-101 image collection is used for measurements. For every single image (total of 9144), the original image and respectively downscaled variations (160x160, 80x80, 40x40, 20x20) are used for querying. To minimize the information loss, the slow but relatively accurate Lanczos 3 down sampling algorithm is used. For each

query, the first 100 hits of the result sets are stored in a database, containing both image id and calculated similarity. Especially the changes in the ranking position of the target image can give some indication of the remaining result quality.

6.1.3. Results

Using the original image for querying in a full retrieval (without restricting the search space by indexes) causes almost perfect results, i.e. the target image is usually the first hit. Yet, it needs to be pointed out that in some cases the features used are not distinctive enough and other images also get the same similarity of a 100% match. This is causing slight but negligible disturbances in the ranking order.

Figure 6.1 provides an overview of all results for the downscaled query images. The x-axis shows the achieved ranking position of the target image. The expected position would always be 1. The logarithmic y-axis displays the normalized average of target images at a given rank. The optimal case would be to have an amount of 1.0 for ranking position 1 without any image in worse ranking positions. Due to the loss of information in the query, several target images gradually drop within the ranking to lower positions, resulting in scatter plots with visible trends.

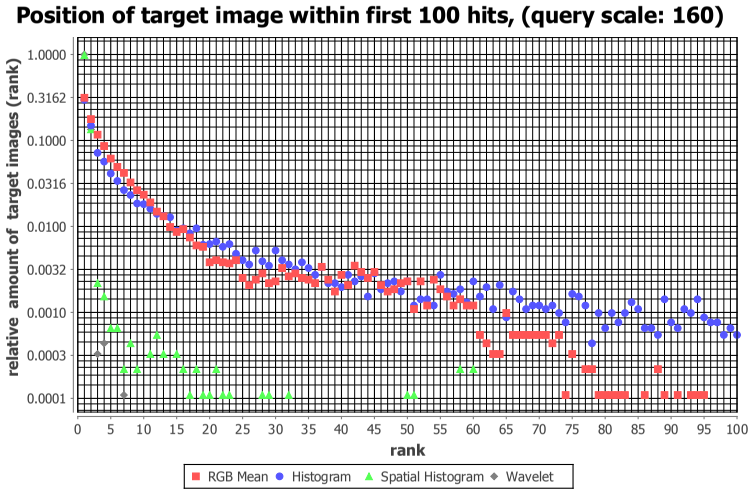
The downscaled 160x160 in fig. 6.1(a) already show a difference to the optimal results. The *Wavelet* feature shows a few outliers, where the original image is ranked slightly lower, but still among the first 10 hits. In comparison, the *Spatial Histogram* feature has much more outliers. The *RGB Mean* and the *Histogram* features perform relatively similar, with a slight advantage of the *Histogram* feature.

The scaling to 80x80 pixels indicates the sensitivity of the *Spatial Histogram* feature to downscaling (fig. 6.1(b)). The ranking quality moves closer towards the values of *RGB Mean* and *Histogram* until it is worse than the *Histogram* results.

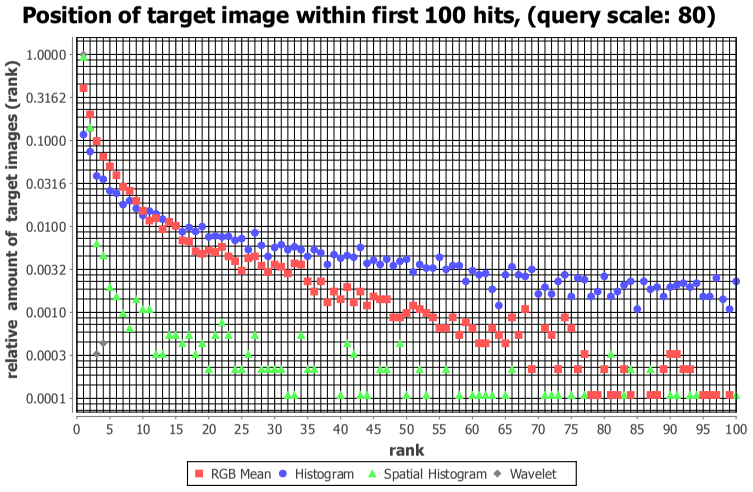
In the 40x40 and 20x20 cases figs. 6.1(c) and 6.1(d), the above trends continue. The peak values of the *RGB Mean* and the *Histogram* features for the highest ranking position keep dropping, especially for the *Histogram* feature. In the end, the *Spatial Histogram* profile is similar to the profile of *RGB Mean*.

6.1.4. Discussion

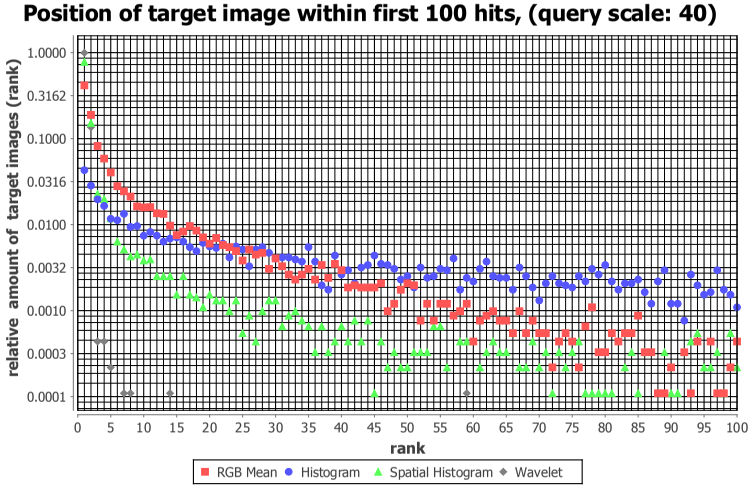
The results show that the *Wavelet* feature seems to be most robust against resizing of the query image. As this is the only tested feature that uses image texture information, it seems that the down sampling preserves texture information better than the colours.



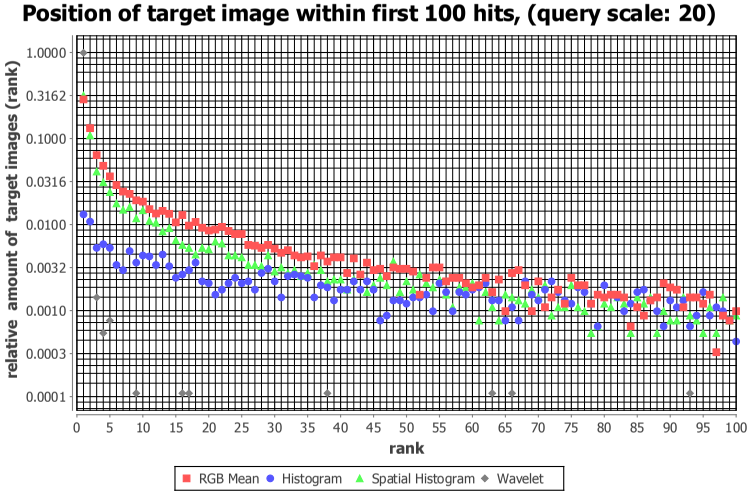
(a) 160x160 query image



(b) 80x80 query image



(c) 40x40 query image



(d) 20x20 query image

Figure 6.1.: Impact of query image sizes

Further, the *Wavelet* feature always uses a downscaled 100x100 pixel image to extract the feature vector instead of using the original image. The *RGB Mean* feature with only 3 moments performs slightly better than the *Histogram* feature which has 12 moments. An explanation for the bad performance of the *Spatial Histogram* feature for 20x20 query images could be the way it is extracting the histograms. By dividing the original image into 4x4 sub images on the 3rd level, each remaining sub image has merely a size of $5 * 5 = 25$ pixels to extract histograms.

6.1.5. Summary

This case study investigated the impact of downscaled QBE images in the retrieval process. The robustness against downscaling depends on the feature vector used. Thus, it is advisable to be careful when attempting to reduce the size of the queries to be transmitted. Otherwise, the loss of information may render the results useless.

6.2. Feature Normalization

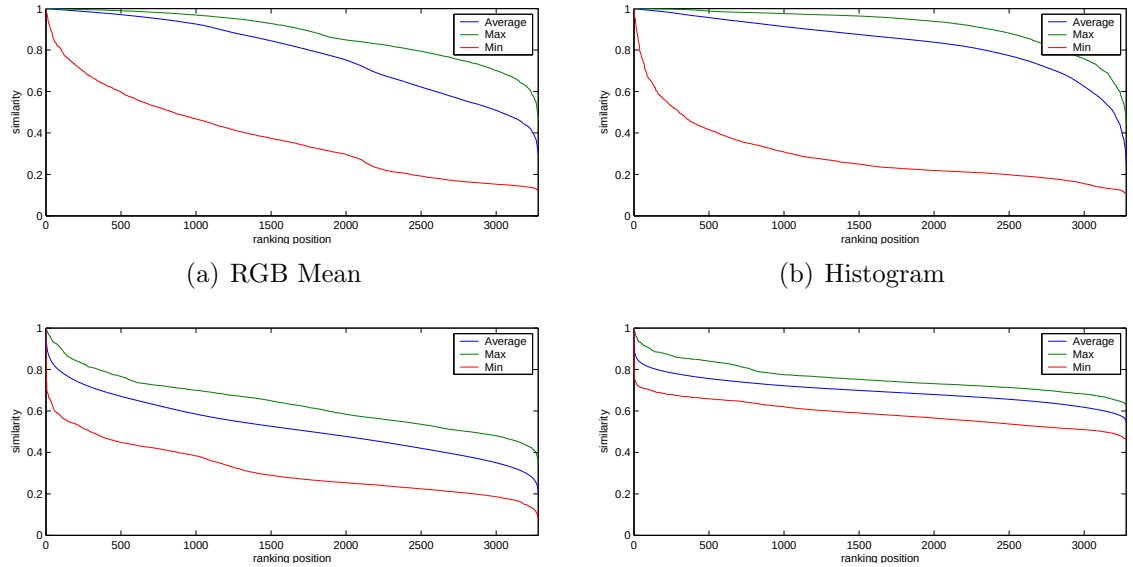
As mentioned in section 3.4.2, each feature vector creates a different similarity curve between 100% match and the least relevant document. Similarities of features are not comparable to other features. For that reason, it is attempted to capture the feature vector specific similarity profiles and use them to normalize the similarity values within each retrieval.

6.2.1. Requirements

- default image database with use-case related content (e.g. photographs)

6.2.2. Testing

First, the original similarity profile of each feature vector plug-in is determined for both image repositories (ETH-80 and Caltech-101). For each ranking position, the average, minimum and maximum value are collected. These profiles are used as reference to create the normalization functions and to measure the changes according to the normalized profiles. Each average curve is split into 20 segments of equal size. The selected boundaries are 1.0 for $rank_0$ and 0.0 for $rank_{|I|+1}$. In between, the average similarity of each



(a) RGB Mean

(b) Histogram

(c) Spatial Histogram

(d) Wavelet

Figure 6.2.: Similarities Cumulated by Rank (ETH-80)

of the 19 equidistant ranking positions are used as reference points for the later normalization. The feature vector modules used: *RGB Mean*, *Histogram*, *Spatial Histogram*, *Wavelet*

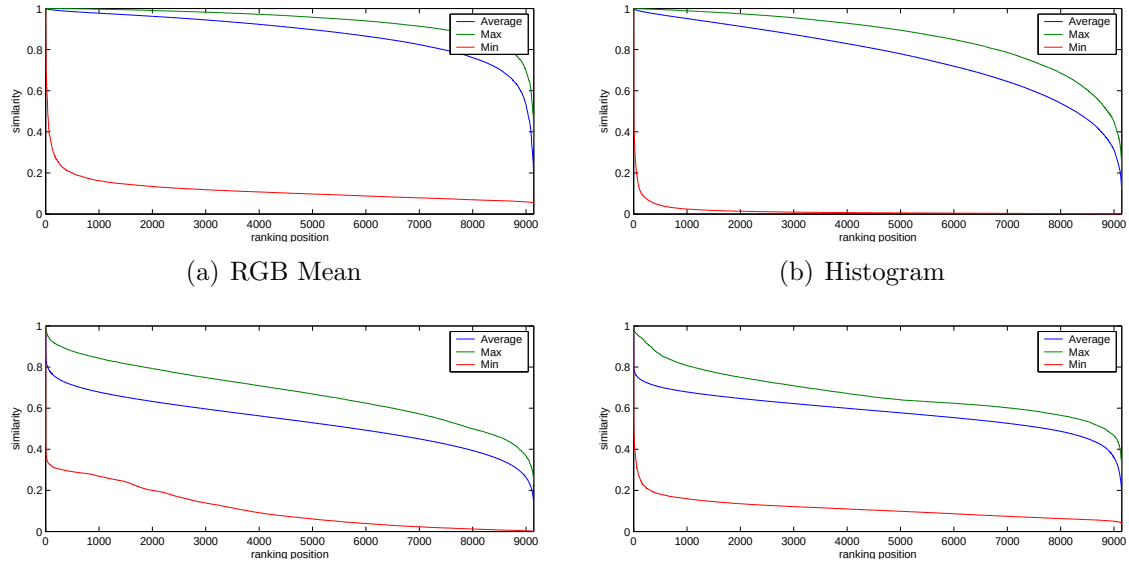
The normalized similarity profiles are expected to be close to the desired function in section 3.4.2.2. Especially for the dataset specific normalizations, the average values should produce only a small error. Normalization parameters extracted from a different image set are expected to result at least in a better profile than without normalization.

6.2.3. Results

This section presents the similarity profiles of the two image collections. In section 6.2.3.1, the original profiles are listed. Section 6.2.3.2 and section 6.2.3.3 contain the normalized profiles according to the original profiles of each collection.

6.2.3.1. Original Profiles

Figures 6.2 and 6.3 show the similarity profiles for the four feature vector plug-ins in the ETH-80 image set containing 3280 images and the Caltech-101 image set containing 9144 images. Comparing the profiles for both datasets shows similar characteristics for each feature. Every single curve starts with a similarity of 1.0 (identity), as each query



(a) RGB Mean (b) Histogram

(c) Spatial Histogram (d) Wavelet

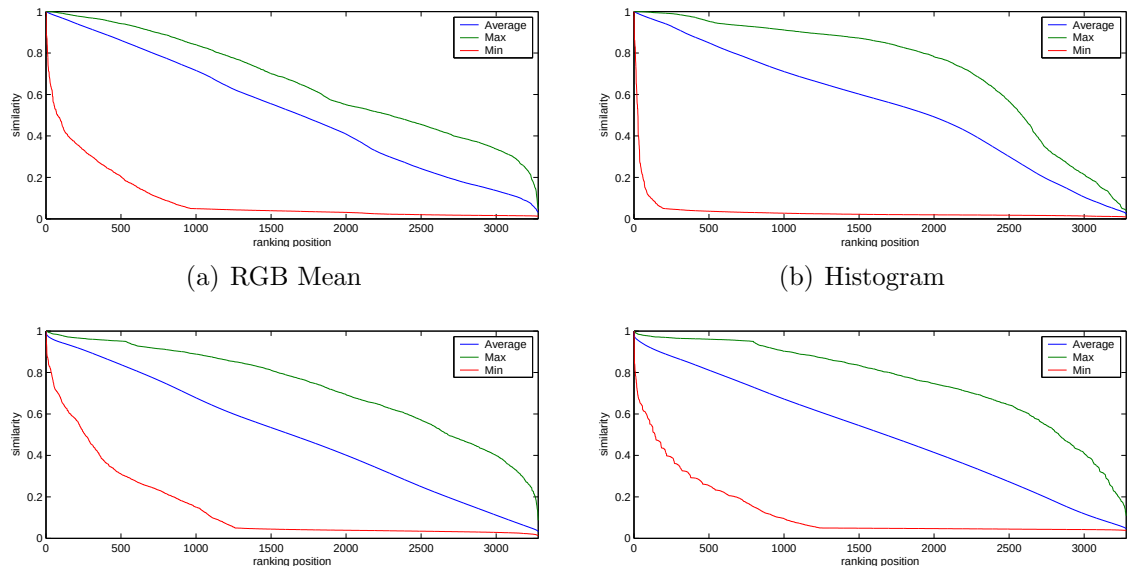
Figure 6.3.: Similarities Cumulated by Rank (Caltech-101)

contains one of the images from the repository. Generally, all profiles are relatively smooth without obvious steps inside.

The *RGB Mean* produces a convex average profile with a relatively shallow decline until about 90% of the repository size. In the final part, the curve drops quickly to similarities near 0.2. The maximum value shows a similar trend, but on a slightly higher level. The minimum value drops quickly in the beginning and then the decline of the slope decreases. In general, the sections near the boundaries show a steep slope, which is even more evident in the Caltech-101 profile (figs. 6.2(a) and 6.3(a)).

The *Histogram* profiles are similar (figs. 6.2(b) and 6.3(b)). In the ETH-80 profile, the decline of the three values is less steep in the central part compared to the ones of *RGB Mean*. In the Caltech-101 profile, the slope of average and max values is steeper than for the *RGB Mean*. Further, the minimum value drops very quickly below 0.1 (at rank 150).

The third pair of profiles shows the behaviour of the *Spatial Histogram* (figs. 6.2(c) and 6.3(c)), forming roughly an "S"-shape. The central section of these profiles is relatively linear (approx. 80%). In contrast to the previous feature vectors, the average value very quickly declines in the first 10% of the profile below a sinilarity of 0.8 before entering the linear section. After reaching a similarity about 0.4, the average similarity starts dropping quickly again. For both data sets, the maximum and minimum profiles



(a) RGB Mean (b) Histogram

(c) Spatial Histogram (d) Wavelet

Figure 6.4.: Similarities Cumulated by Rank (ETH-80, normalized by ETH-80)

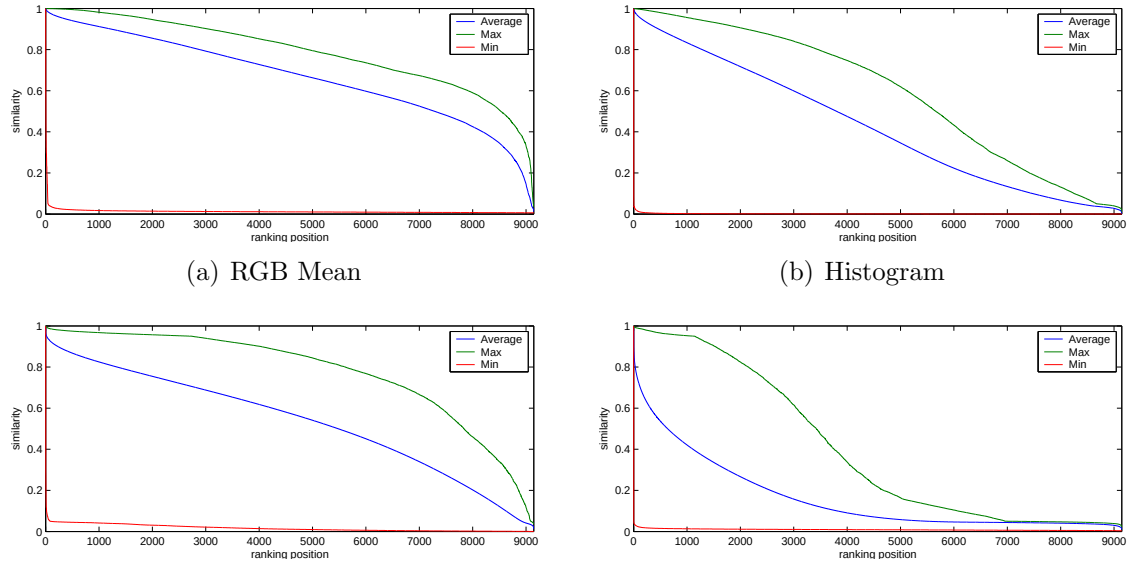
are roughly of the same shape as the average profile, whereas the extreme values of the Caltech-101 dataset are further apart.

The last feature vector to be tested is the *Wavelet*. Again, the profiles resemble an "S" (figs. 6.2(d) and 6.3(d)). Compared to the *Spatial Histogram*, the slope of the central section is shallower. For the ETH-80, most values lie in the corridor with a similarity between 0.6 and 0.8 and for the Caltech-101, the corridor is between 0.7 and 0.5. The maximum values are mostly 0.1 above the average values. For the minimum values, the ETH-80 profile is not much below the average profile, but on the Caltech-101 dataset, the gap between average and minimum values is about 0.5

6.2.3.2. Normalized by ETH-80

In figs. 6.4 and 6.5, the similarity profiles are normalized according to the normalization function described in section 3.4.2.2. The parameters used are extracted from the original similarity profile of the ETH-80 dataset.

As intended, the average values of the normalized ETH-80 (fig. 6.4) profile form a relatively straight line from ($rank_1$/similarity 1.0) down to ($rank_{|I|}$/similarity 0.0). Except from a slightly too quick drop near the first ranks and some minor bulges in the central section for *RGB Mean* (fig. 6.4(a)) and *Histogram* (fig. 6.4(b)), the normalized average values are close to the desired shape. In the central section, the extreme values



(a) RGB Mean (b) Histogram

(c) Spatial Histogram (d) Wavelet

Figure 6.5.: Similarities Cumulated by Rank (Caltech-101, normalized by ETH-80)

are further apart from the averages than in the original profile.

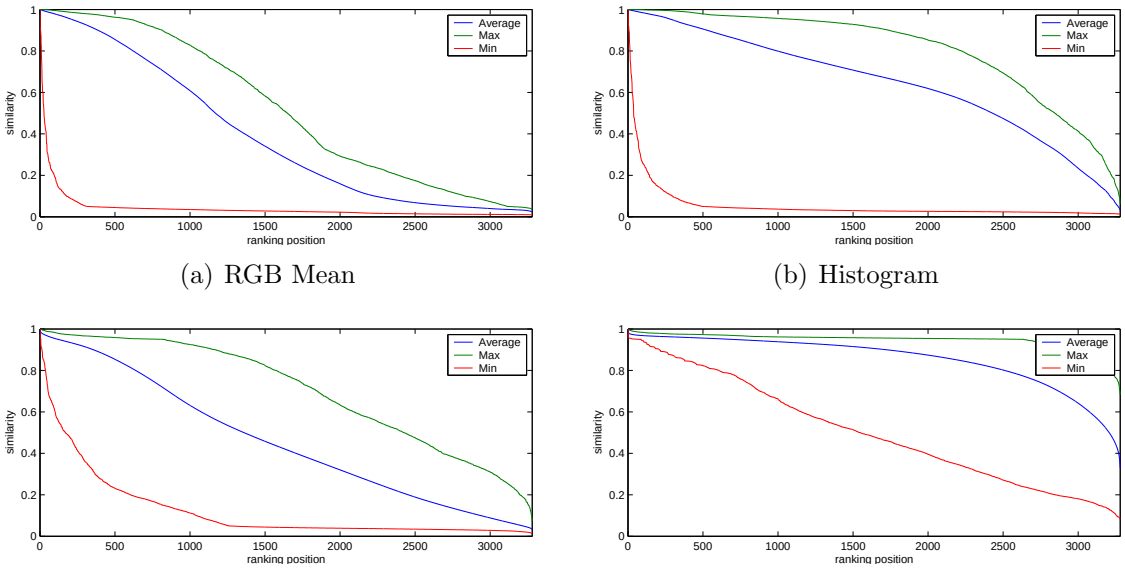
The same normalization applied to the Caltech-101 image dataset is much less fitting the desired function fig. 6.5. Nevertheless, at least the actively used similarity range is clearly more evenly distributed between 1.0 and 0.0 than before (fig. 6.3). The average values of three profiles are closer to the intended function, but in the case of the *Wavelet* feature (fig. 6.5(d)), the profile is too steep in the beginning and too shallow in the second half. In all four profiles, the minimum drops to similarities below 0.1 very quickly.

6.2.3.3. Normalized by Caltech 101

In figs. 6.6 and 6.7, the similarity profiles are also normalized according to the normalization function described in section 3.4.2.2. The parameters used are extracted from the original similarity profile of the Caltech-101 dataset.

Applying the Caltech-101 normalization on the ETH-80 images generates profile not quite fitting the desired function (fig. 6.6). The average similarities take advantage of the full range from 0.0 to 1.0, but the curves are much less straight than in the ETH-80 normalized casefig. 6.4. Especially the *Wavelet* feature profile is not close to the intended function, but lies far above it (fig. 6.6(d)). The average similarity for $rank_{|I|}$ is 0.33.

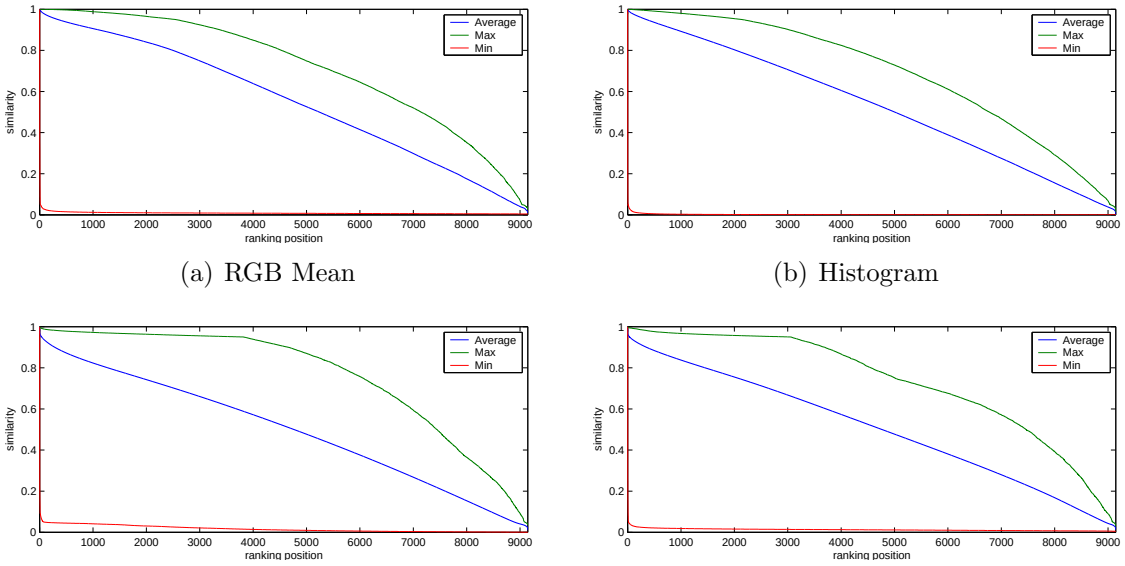
As intended, the average values of the normalized Caltech-101 (fig. 6.7) profile form a relatively straight line from ($rank_1$/similarity 1.0) down to ($rank_{|I|}$/similarity 0.0). The



(a) RGB Mean (b) Histogram

(c) Spatial Histogram (d) Wavelet

Figure 6.6.: Similarities Cumulated by Rank (ETH-80, normalized by Caltech-101)



(a) RGB Mean (b) Histogram

(c) Spatial Histogram (d) Wavelet

Figure 6.7.: Similarities Cumulated by Rank (Caltech-101, normalized by Caltech-101)

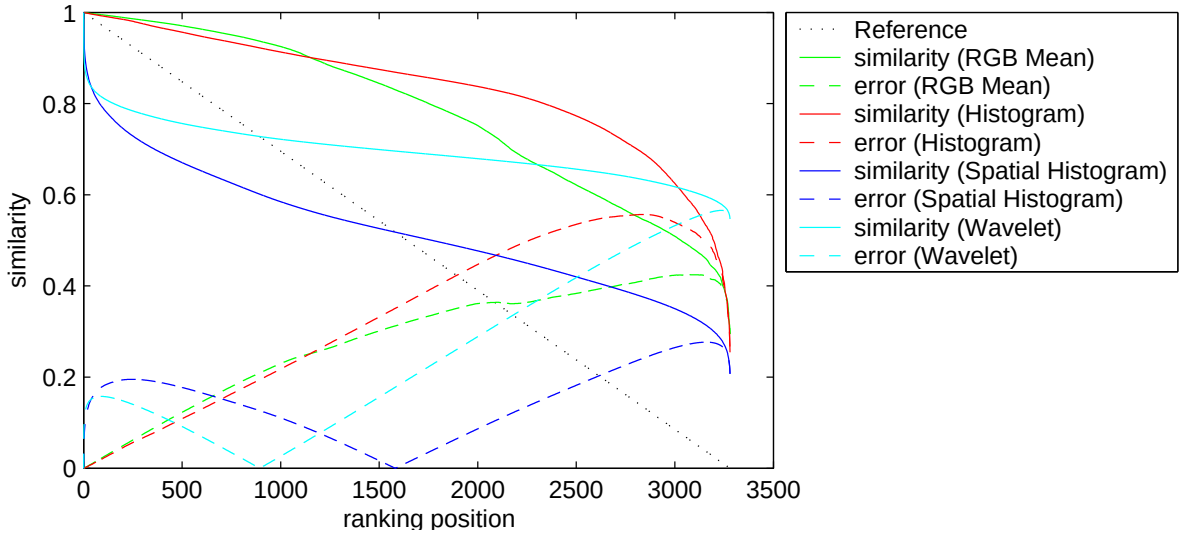


Figure 6.8.: Error of Original Similarity Profiles for ETH-80 Images

most prominent deviations are the initial quick drop to similarities around 0.95 for the highest ranks. The maximum values do not show any special behaviour. Again, the minimum values drop extremely quick below a similarity of 0.1.

6.2.4. Analysis

Below, the results of the testing procedure are being analyzed. Especially the deviation from the reference function is is calculated to determine the individual error for each test case. The figures below plot the four similarity profiles of the feature vectors and their absolute error compared to the reference function (dotted line). In the summary (section 6.2.4.4), the errors of all features are compared to each other.

The error ϵ for each rank and the cumulated error E are defined as

$$\epsilon_{\text{rank}} = \left| \left(1 - \frac{\text{rank}}{|I|}\right) - s(\text{rank}) \right| \tag{6.1}$$

$$E = \sum \epsilon_{\text{rank}} \tag{6.2}$$

where $s(\text{rank})$ is the average similarity of all results at position rank.

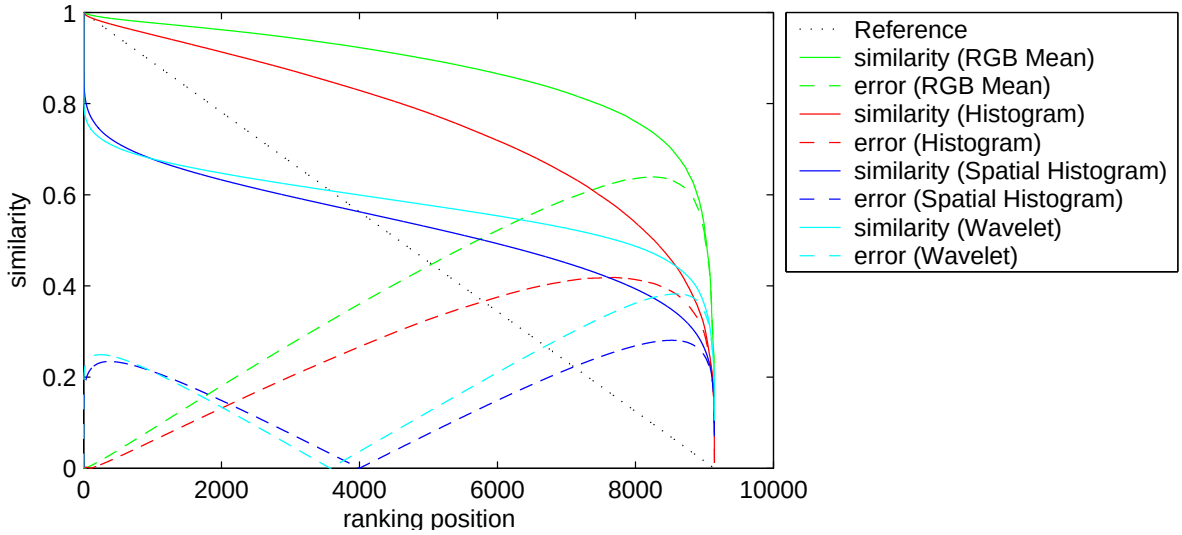


Figure 6.9.: Error of Original Similarity Profiles for Caltech-101 Images

6.2.4.1. Original Profiles

Figure 6.8 contains all similarity profiles for the average values for the ETH-80 image repository. In the range up to approximately rank 400 ($\approx$ 12% of $|I|$), the *Spatial Histogram* and *Wavelet* features reach an error of about 0.2 because of their initial steep drop. With a growing rank, the error drops to 0 and then rises up to 0.57 (W) and 0.277 (SH). The error of the other features *RGB Mean* and *Histogram* ascend much slower and peak in the less relevant ranks around 3000 ($\approx$ 90% of $|I|$).

In fig. 6.9, all similarity profiles for the average of the Caltech-101 repository are contained. The error graphs have comparable characteristics to the previous profiles. Below the 2000 ($\approx$ 22% of $|I|$) *Spatial Histogram* and *Wavelet* both show their first error peak above 0.2 and their second one around rank 8500 ($\approx$ 93% of $|I|$). Again, the error of the other features *RGB Mean* and *Histogram* ascend much slower and peak in the less relevant ranks around 8500.

6.2.4.2. Normalized by ETH-80

In fig. 6.10, the error for the optimized ETH-80 dataset is depicted. It shows a very low error for all four features, whereas the *Histogram* feature vector shows a small error bulge around rank 2000 ($\approx$ 60% of $|I|$) with a peak deviation of 0.1.

Figure 6.11 is based on the Caltech-101 images, optimized by the ETH-80 parameters. Most features have an error below 0.1 for the first 2000 ($\approx$ 22% of $|I|$) ranked results. On later ranking positions, the error of the *RGB Mean* feature keeps increasing, while

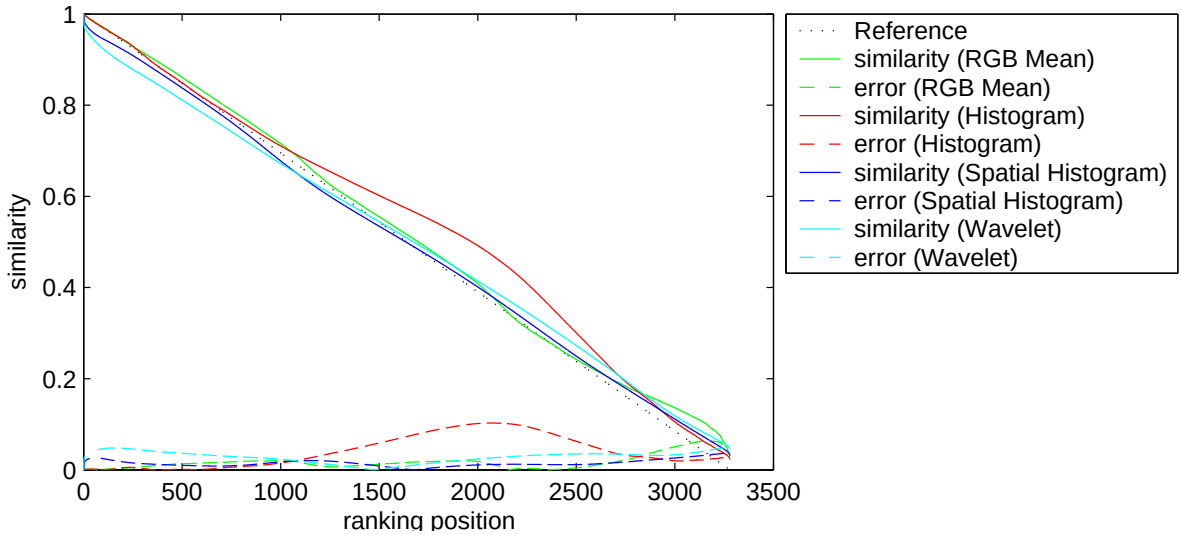


Figure 6.10.: Error of Normalized (ETH-80) Similarity Profiles for ETH-80 Images

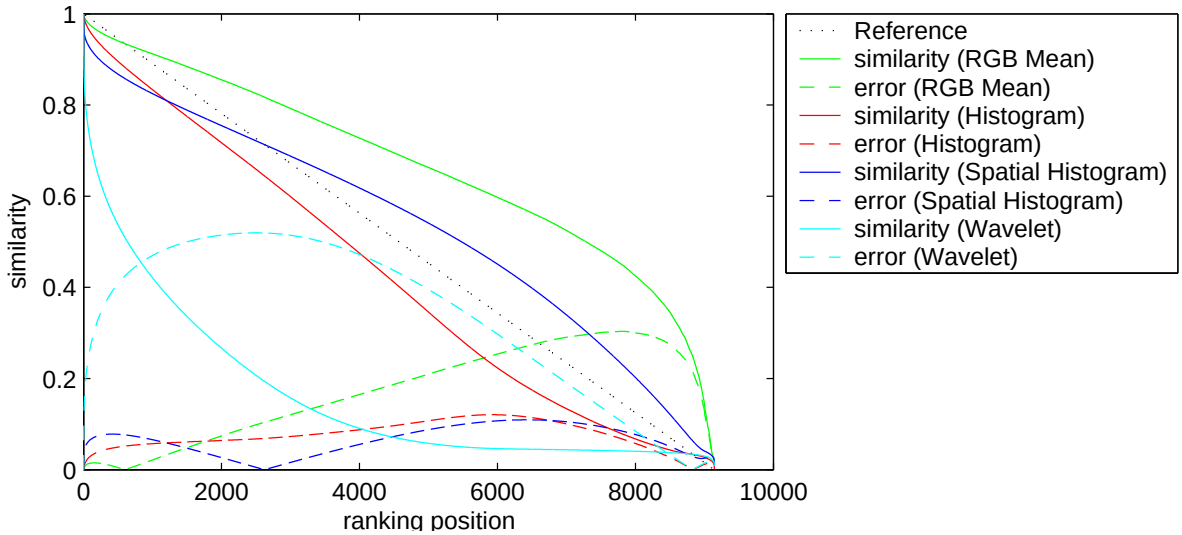


Figure 6.11.: Error of Normalized (ETH-80) Similarity Profiles for Caltech-101 Images

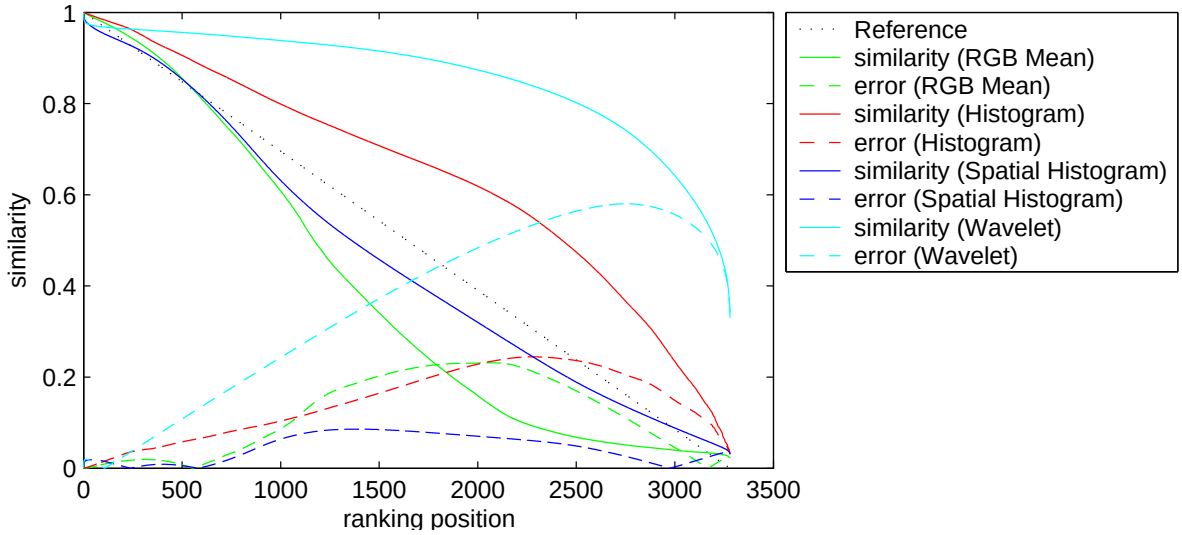


Figure 6.12.: Error of Normalized (Caltech-101) Similarity Profiles for ETH-80 Images

Histogram and *Spatial Histogram* remain below 0.1. The worst performing feature is the *Wavelet*. At rank 2000, the error peaks at about 0.5 and then decreases again.

6.2.4.3. Normalized by Caltech 101

In the opposite way — the ETH-80 images profiled with the Caltech-101 function (fig. 6.12) — the results are less convincing than in fig. 6.11. While the *Spatial Histogram* always remains below 0.1, the *RGB Mean* and *Histogram* exceed the 0.2 in the second half of the results. Again, the worst normalized feature vector is the *Wavelet*. Opposed to the ETH-80 normalized Caltech-101 profile, the average similarity slope of the *Wavelet* feature is too shallow in the first half.

In fig. 6.13, the error for the optimized Caltech-101 dataset are plotted. As intended, the error for all four features is very low and always remains below 0.1. The strongest deviations can be seen for the *Spatial Histogram* and *Wavelet* features in the first 2000 results ($\approx 22\%$ of $|I|$) where the average similarity is below the reference.

6.2.4.4. Summary

A concise overview of the results described above is given in table 6.1 and fig. 6.14. In most test cases, the normalized similarities push the total error E below 50% of the original results. The only exception to this rule are the two *Wavelet* based errors while using the normalization parameters from the opposing image repository. If the matching normalisation parameters are chosen, E always remains below 0.056 (highest value for

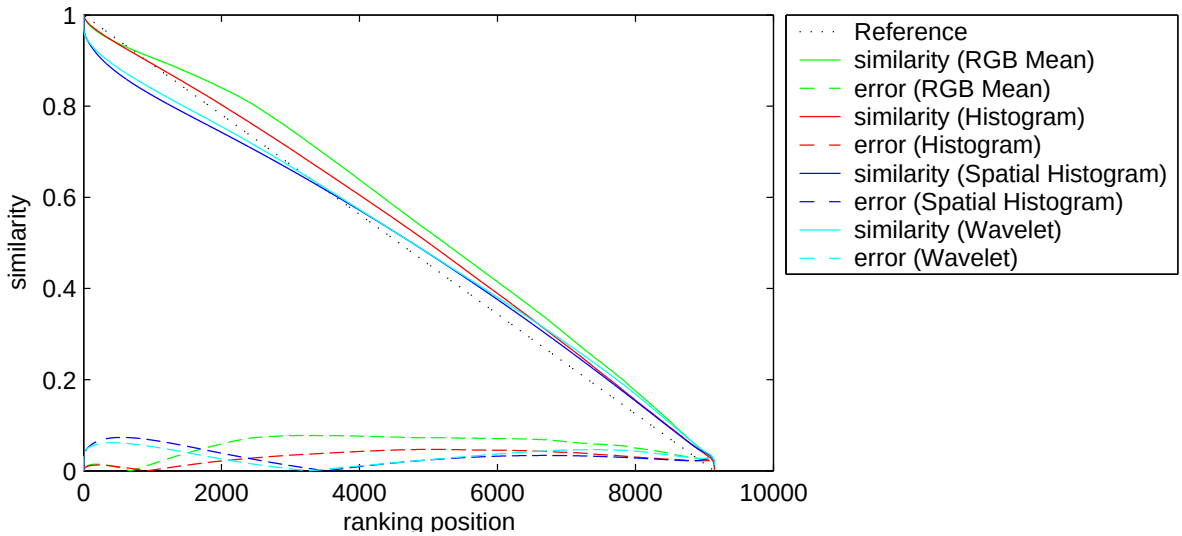


Figure 6.13.: Error of Normalized (Caltech-101) Similarity Profiles for Caltech-101 Images

Table 6.1.: Cumulated Average Error

dataset	feature vector	original	normalized ETH-80	normalized Caltech-101
ETH-80	*RGB Mean*	0.28085	0.0173	0.11572
	Histogram	0.33513	0.040525	0.14398
	Spatial Histogram	0.14362	0.014856	0.044396
	Wavelet	0.24816	0.028131	0.35692
Caltech-101	*RGB Mean*	0.37754	0.055727	0.17042
	Histogram	0.26203	0.031286	0.075503
	Spatial Histogram	0.15737	0.03166	0.064893
	Wavelet	0.19342	0.031768	0.33909

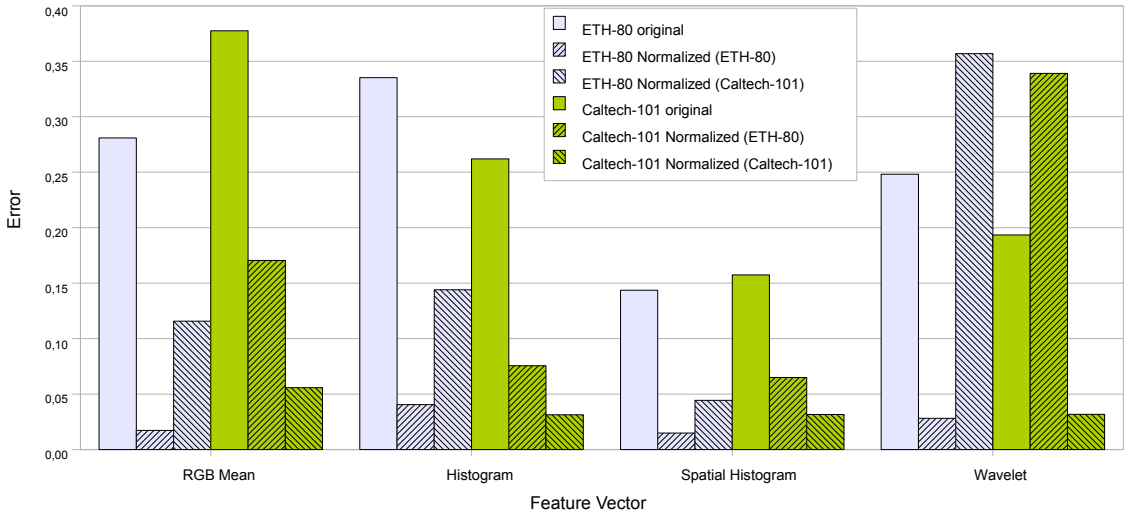


Figure 6.14.: Cumulated Average Error

Caltech-101, *RGB Mean*). Using the "wrong" parameters, the best performing features generate an error twice above the optimized ones. The best improvement has been achieved for the *RGB Mean* feature in the ETH-80 dataset where the error is reduced to 6% of the original error.

6.2.5. Discussion

Generally, the expectations were satisfied by the test. Using database specific normalization parameters, the total error could be reduced in every case. This leads to much easier comparable feature vectors. The normalized similarity value between query and repository image is now roughly indicating the relative position of the image in the ranking independently from the feature vector used.

Using the normalization parameters of a different image set worked surprisingly well. At least, the similarity range $[0.0, 1.0]$ is used more evenly than before. The only real exception is the *Wavelet* feature. The main reason for this behaviour lies in the similarity calculation that returns values above 0.5 in almost any case (see figs. 6.2(d) and 6.3(d)). This causes the normalization function to be inaccurate for values between 0.0 and 0.5 where in this case $\frac{1}{2} = 50\%$ of the original range are mapped to only $\frac{1}{20} = 5\%$ of the normalized range. Also, the lower boundary for $rank_{|I|+1}$ is arbitrarily mapped to 0.0 even if the similarity between two feature vectors may never be close to this value.

Another problem becomes evident in the highest ranks. Especially the features with higher dimensionality (i.e. *Spatial Histogram* and *Wavelet*) tend to drop too quickly at

high similarities. A finer granularity of the normalization function in this range might reduce this effect. This issue also raises the question, if it is necessary to normalize the similarities at less relevant ranks. A retrieval system is supposed to return a relatively small set of highly relevant results rather than a fully sorted list of all known documents. Even for the highly unclear boundaries of CBIR, it usually makes no sense to return everything. Instead of separating the whole range into equi-distant segments to determine the normalization function, another approach may be more useful. One possible approach would be a logarithmic segmentation by recursive binary splitting of the respective "upper" half of the ranking. This would easily scale to large databases without losing accuracy or storing too many normalization points. Also the error function should treat deviation in the highly relevant sections more than in the less relevant ones.

The minimum values have a tendency to be extremely low, especially in the Caltech-101 dataset (fig. 6.3). This can be explained by a collection containing at least one image having no similarity to any other image. As the minimum profile is generated by using each image of the repository, such an outlier far away from any cluster of similar images easily pulls the boundary down. Thus, the minimum values don't have a very high expressiveness for this case study except from showing the extremes. More useful for future testing could be a set of given confidence intervals to determine statistical boundaries of how many similarity values are below or above a given threshold.

6.2.6. Summary

The above testing procedure can be applied during the development of new feature vectors. In multi-feature applications, the similarity function for each pair of vectors may not easily be constructed to generate homogeneous distribution of similarities. Generating a set of normalization parameters would simplify this task and it would also allow for a use case specific optimization.

6.3. Estimating the Improvement Capabilities of Different Features

This section evaluates the estimated improvement capabilities of each implemented feature vector. It is calculated, by which factor a feature vector can improve the retrieval performance for various categories. This information is a valuable help to select a certain feature vector to be used for searching a category.

6.3.1. Requirements

- default image database with annotated categories

6.3.2. Data Preparation

The feature vectors used have various levels of complexity. The simplest one is a trivial RGB mean value, previously used for tests in a query language [96] (*mean*). Much more sophisticated is the use of 12 stochastic moments from the image histograms by Al-Omari and Al-Jarrah [4] (*stochastic2*). The same feature is also used in feature vector with added spatial awareness, by splitting the original image into a quad tree of histograms [92] (*stoch_quad*). Finally, a wavelet based feature is used [58] (*wavelet*). It is expected that the measured quality of the four feature vectors is related to the complexity level.

These feature vectors are applied on the Caltech 101 image repository [40]. This collection consists of 101 categories with $|I| = 9144$ images, each one depicting a specific object. The images are reasonably well normalised by scaling, centring and rotating the objects into comparable positions. A set of more or less random background images is also included.

The testing has been carried out on a PC with "Intel Core2 Duo" CPU (2 * 2.33 GHz). A data collection run took approximately 4 hours. During this time, no speed-optimizing indexing has been used by the CBIR search engine and each query has been compared to every available image in the repository. Thus, the retrieval results represent the "optimal" quality without any loss caused by indexing techniques. Further, this case study does not use the normalized similarities.

6.3.3. Data Collection

For the benchmark preparation, the whole repository is imported into the CBIR system, extracting the image features and the category keywords. Then every single image feature is used as a query and the first 50 ranked results of each retrieval are stored in a database. Among the first 50 hits should be as many related images from the same category as possible. The higher the recall for the target category, the higher the measured quality of the feature vector used.

6.3.4. Data Normalization

A known issue of the Caltech 101 dataset is the strong variation of category sizes $|R_{category}|$. Some of which are under-represented with only as much as 31 images and others contain 800 samples. Therefore, the collected data needs to be normalized.

First of all, the under-represented images have to be adjusted to the other categories. This is done by calculating a normalized recall value $0 \leq \rho \leq 1$ for the first 50 results:

$$\rho_{normalized} = \frac{tp}{\min\{|R_{category}|; 50\}} \tag{6.3}$$

The second necessary step is the normalization of these recall values according to the size of each category set. The random precision π_{random} of picking an image of the required category depends both on the repository size n and the amount of samples for a category $|R_{category}|$:

$$\pi_{random} = \frac{tp + fn}{tp + tn + fp + fn} = \frac{|R_{category}|}{|I|} \tag{6.4}$$

With this data, the result quality of every single query can be directly compared to random picking, calculating an improvement factor τ.

$$\tau = \frac{\rho_{normalized}}{\pi_{random}} \tag{6.5}$$

Values below 1 should never occur in realistic scenarios, as this would indicate that randomly choosing images from the repository performs better than using the benchmarked feature vector. The higher the improvement factor, the better. For each category, the averages and the standard deviation of the improvement factor are calculated.

6.3.5. Results

The findings of testing the 9144 images in 102 categories are visualized in fig. 6.15 and summarized in table 6.2. Also, a couple of sample images from various categories are presented (figs. 6.16 - 6.20).

Figure 6.15 visualizes the strength of every feature to recognize a specific category. Each bar represents a single category, sorted alphabetically by the given folder name of the collection. The height of a bar represents the average improvement τ that has been achieved by using the corresponding feature vector. The standard deviation is indicated by a vertical H-shaped line on top of each bar.

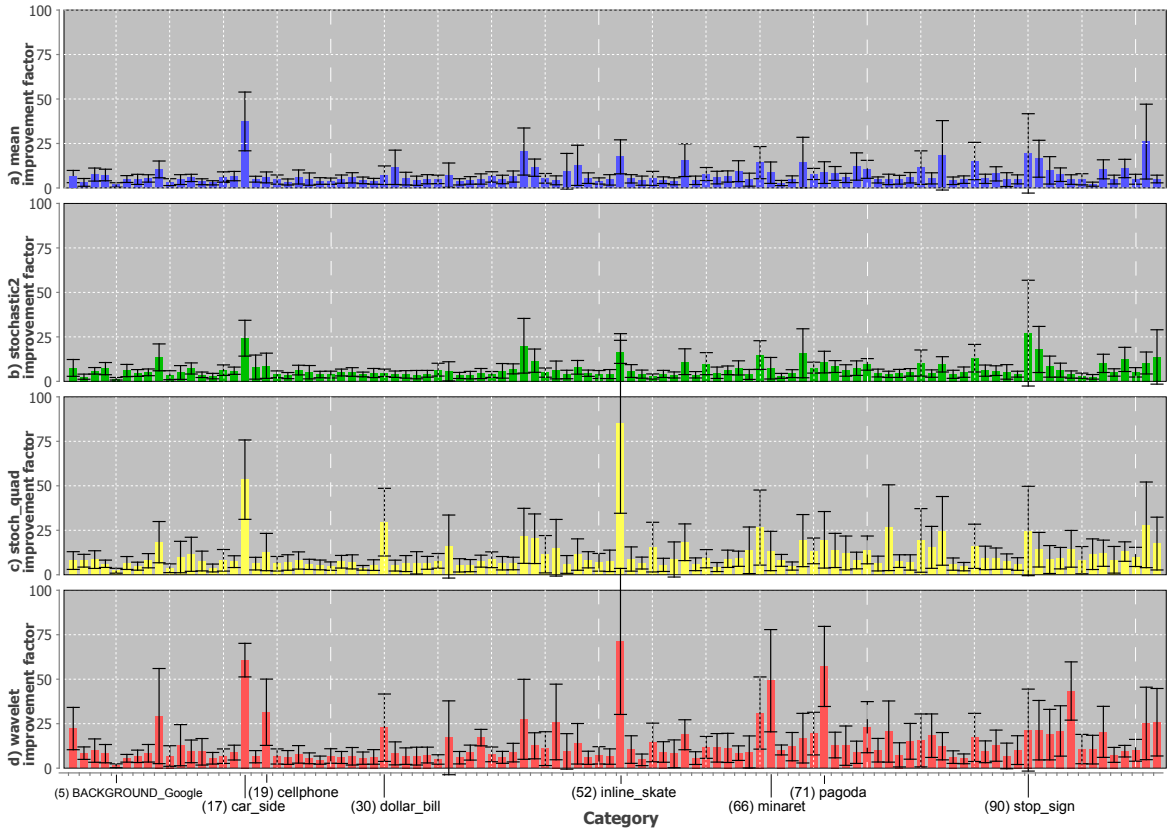


Figure 6.15.: Improvement Factor

6.3.6. Analysis

Table 6.2.: Summarized Improvement

Feature	Average				Standard Deviation		
	avg	*min*	*max*	$avg \geq 10$	*avg*	*min*	*max*
mean	7.37	1.69	37.4	20	4.51	1.14	22.3
stochastic2	6.81	1.46	27.11	17	4.01	0.72	29.73
stoch_quad	11.57	2.25	85.04	37	7.89	1.32	50.48
wavelet	14.0	1.34	71.23	50	9.07	1.09	41.04

The results for the 102 categories is summarized even further in table 6.2. It compares some highly condensed characteristics of each feature vector, i.e. the *average*, *minimum* and *maximum* for both improvement factor average and standard deviation.

In many cases the average improvement factor is close to 5. For some categories, the average value ranges from 10 to 85. These peaks depend on the feature vector used and do not occur in every diagram equally. The amount of peaks is relatively low with the *mean* (fig. 6.15 a) and the *stochastic2* (fig. 6.15 b) feature vector. The number increases with the *stoch_quad* (fig. 6.15 c) and *wavelet* (fig. 6.15 d) feature vectors. The standard deviation of each category/feature pair varies from about 10% of the average to more than 100%. The feature vectors used do not seem to have a large impact on this behaviour and the standard deviation seems to be closely related to the average with a factor of $\frac{average}{standard\,deviation} \approx 1.5$.

In comparison, the average improvement of the more sophisticated feature vectors (*wavelet: 14.0*, *stoch_quad: 11.57*) is by a factor of 2 higher than that of the simpler ones (*mean: 7.37*, *stochastic2: 6.81*) in relation to a random result. The amount of categories above a threshold of 10 shows a similar tendency. According to these numbers, the histogram based feature vector *stochastic2* with 12 parameters seems to perform slightly worse than the trivial RBG *mean* with only 3 parameters, which is an unexpected outcome.

(a) (b) (c) (d) (e) (f)

Figure 6.16.: Caltech 101 - category subset: BACKGROUND_Google

The worst performing category is the "BACKGROUND_Google" (fig. 6.16). The average improvement factor of all four feature vectors represents the minimum in table 6.2. All four values lie very close to 1. This means, there is almost no improvement compared to random picking at all. As this specific category is defined by being an unsorted collection of random images with no particular semantics, the algorithms cannot find any useful information that links any two of these images together.

(a) (b) (c) (d) (e) (f)

Figure 6.17.: Caltech 101 - category subset: car_side

One of the best recognized categories is the "car_side" (fig. 6.17). With some knowledge about the contained images and the feature vectors used, the reason becomes obvious immediately. Every feature vector used in this investigation is analysing the image colours and this category is the only one which consists completely of gray scale images. Thus, each feature vector is able to cope with it very well.

(a) (b) (c) (d) (e) (f)

Figure 6.18.: Caltech 101 - category subset: inline_skate

The most prominent feature-dependent results are found in the category "inline_skate" (fig. 6.18). This is probably due to the fact that the samples this category are very homogeneous. With only 31 images, this is one of the smallest categories, but the object, background and orientation of most representations is very similar (mostly gray, white, front points to the left). While the purely colour based feature vectors generate reasonable results, the ones which consider spatial information too are capable of retrieving most related images directly (*stoch_quad: 85.04*, *wavelet: 71.23*). The large standard deviation is caused by the few outliers, where background or orientation differ. Some other categories, such as "cellphone", "minaret" and "pagoda" benefit also from the spatial information and especially very similar backgrounds.

A category with a very high standard deviation is the "stop_sign" (fig. 6.19). This set

(a) (b) (c) (d) (e) (f)

Figure 6.19.: Caltech 101 - category subset: stop_sign

contains 64 samples of an highly iconic motive. All over the world, stop signs look almost identical, especially according to shape and colours. Approximately half of the samples show a stop sign with only very small portions of background (e.g. figs. 6.19(a), 6.19(c), 6.19(e), 6.19(f)). These images form a very compact cluster in th feature space. On the other hand, several images show the sign embedded in scenery or from an angle. These samples form a less defined cluster, where the feature vectors have far less distinction to samples from other categories.

In general, higher bars indicate a better performing feature vector for a specific category. Also, the standard deviation should be relatively low. In that case, a CBIR system would return a large set of highly similar images for every existing query image for a category. This benchmark approach could be beneficial to compare the performance of various feature vectors and for fine tuning new ones. It compensates most of the imbalance in category sizes and also takes the repository size into account. It also indicates, which image categories can be handled easily and where the feature vector specific problems lie. The further summarized parameters (table 6.2) can be used as a more general benchmark.

6.3.7. Limitations

This case study uses the well known image Caltech 101 repository [40], which has certain weaknesses. Due to limited resources, the investigation did use the complete image to extract a feature vector instead of applying a sophisticated regional clustering algorithm. Thus, the background of the images may have a strong influence on the results.

(a) (b) (c) (d) (e) (f)

Figure 6.20.: Caltech 101 - category subset: airplanes

Also, some categories contain several sub-categories of very similar objects. The "airplanes" (fig. 6.20) with 800 samples contains military jets, commercial and private planes of all sizes on different backgrounds (e.g. sky, grass, concrete). These sub-categories may cause a very strong recall among each other, but other semantically related images could not be retrieved, causing a low average improvement. Very small classes with a homogeneous image set easily achieve a much better recall . Picking a more homogeneous repository would probably help to overcome these fundamental issues.

The amount of feature vectors tested was relatively small. Each one used colour information, but not a single one is capable of extracting and matching certain objects and their shapes. In future benchmarks, additional feature vector implementations are going to be tested and compared.

As the benchmark performs a retrieval query with every single image in the repository and for every feature vector to be assessed, this benchmarking approach does not scale very well. Without further optimizations, the run time is at least linearly increasing with the repository size. If non-ideal results are also acceptable, the use of dedicated indexing structures for each feature vector could largely increase the scalability of this approach.

6.3.8. Discussion

The case study shows up a way of comparing several feature vectors according to their classification abilities. It was attempted to minimize the influence of the inhomogeneous Caltech 101 image collection [40] in the results. The final normalized data indicated that the complex *stoch_quad* and *wavelet* features are suitable to retrieve several image categories in this repository very well. Other categories may require another feature vector or are simply too inhomogeneous to be retrieved. The benchmark provides feedback of how homogeneous the samples of a category are. In some cases, it might be beneficial to split a category into more detailed sub parts with additional semantic value.

6.3.9. Summary

The average provides evidence of how well a given category can be clustered by using the feature vector and its similarity function. The average improvement factor ranges from 1.34 up to 71.23 (both for the wavelet feature). In this case study, a suitable threshold seems to be 10. The total amount of categories above this threshold correlates with the average improvement factor and is another quality indicator.

The standard deviation visualizes, how the clusters of highly similar images are composed. A low deviation (i.e. $\lessapprox 2$) indicates clusters of similar size. A high deviation (i.e. $\gtrapprox 10$) in contrast indicates clusters of unequal size. This can be a problem, if for example single images of a category contain a very special representation. Then these special instances cannot be retrieved easily with the system.

6.4. Supervised Learning

In this section, the learning algorithm described in section 3.7.7 is evaluated. This section mentions some results published in a recent conference paper [91].

The testing has been carried out on a Solaris 10 x86 server system with 4 CPU-cores used in parallel.

6.4.1. Requirements

- Image set of annotated categories

6.4.2. Preparation & Implementation

The image collection used is the ETH-80 collection from the ETH CogVis project [67, 68]. It contains in total 3280 images of 8 different annotated object categories. Each category contains 10 different objects and have an equal size of 410 images. Each object is available from several angles and is located in front of a blue background. Additional data about the object boundaries are also available from black and white mapping files. This collection is sufficient to test the already available feature vector modules without the additional need of segmentation algorithms. Further advantages of using this image set are:

1. Each image represents a single object/concept
2. The object/concept to be learned fills most of the image area, is centred and completely visible
3. The ETH-80 collection provides additional mask files which define the relevant segment

In this case study, a set of relatively simple feature vectors is used, which are described in [96]. Both the RGB Mean (RGB) and the Histogram (H) module are also extended

and only use the relevant pixels defined by the provided map files. Both Spatial Histogram (SH) and Wavelet (W) remain unchanged. They are expected to be sufficient for evaluating the learning algorithm, as they usually return better results than random guessing (see. [113]). More advanced feature vectors are currently not available for testing, but could easily be added to the system in the future, as this learning approach allows for modular extensions.

Prior to running the learning algorithm, all images were imported into the CBIR repository. At that stage, the time consuming feature extraction takes part once. The remaining computation then relies on these stored feature vectors. To eliminate the influence of non-optimal indexing structures, a full scan over all existing feature vectors is performed for each single retrieval step.

For each of the eight categories, the given 410 images are used as training set. To prevent over fitting, the algorithm is manually limited to a maximum of 20 SHOULD clauses and at most 3 additional MUST/MUST_NOT clauses for each positive one.

6.4.3. Testing

The testing process is split into several steps, adding more detail information each time. This is achieved by enabling more options in the learning algorithm.

1. Single Clauses
2. Multiple SHOULD Clauses
3. Additional MUST/MUST_NOT Clauses
4. Additional Boost Parameter

Every test run generates a query for each category. These queries are then analyzed in detail by calculating the results they produce in the CBIR environment. For those results, Precision/Recall diagrams are generated. The result set size in the diagrams is normalized by the category size 410 and in some cases by twice that size (820) to the range $[0.0, 1.0]$. Results above that size are not considered to be of high interest. The optimal retrieval would achieve a precision of $\pi = 1.0$ among the first 410 images and a precision of $\pi = 0.5$ for this result size is assumed to be reasonable in difficult cases. A searcher should browse at least these results in order to find a suitable image, simply because the repository contains that much relevant images. After this threshold, determined users should also find more relevant hits. Thus, the recall ρ should still

increase, even though the precision is less crucial. If the precision is not very good for the minimum result size of 410, it is aimed to achieve an improved recall until 820.

The results of each test run are used to fine tune some learning parameters, such as the α of the F-Measure and combination rules. These parameters are also used in the subsequent runs. Further, this case study does not use the normalized similarities.

6.4.4. Results

6.4.4.1. Effect of Single Clauses

As already discussed in section 6.3, each feature vector has unique strengths and weaknesses. For each category, the best available feature vector may vary.

The precision-recall diagrams in fig. 6.21 visualize the discriminative power of the single best feature vector found for querying each category. It is a first indication of how well a category can be learned with the available feature vectors. The categories "apple", "pear" and "tomato" (fig. 6.21(a), fig. 6.21(g), fig. 6.21(h)) indicate a high achievable precision with short queries. As the category "cow" performs worst, it is used for assessing the optimization steps.

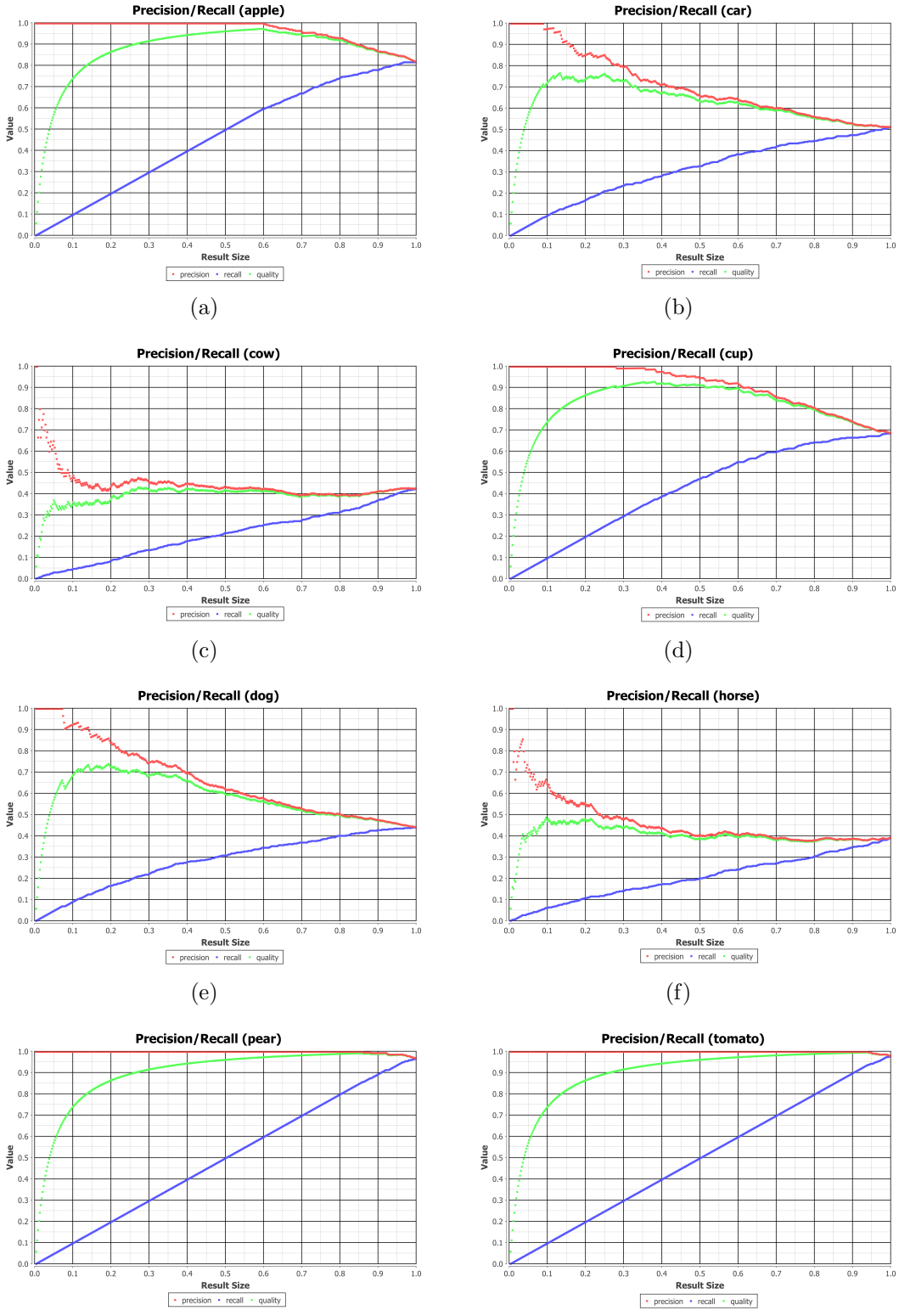
The F-Measure parameter $\alpha = 0.2$ appears to be a reasonable trade-off between precision and recall. By favouring the precision, the learning algorithm is forced to find more restrictive thresholds, filtering out more irrelevant results. This allows more complex queries to focus on multiple highly relevant clusters instead of trying to fit as much as possible into a single clause.

6.4.4.2. Effect of Additional Clauses and Tolerance

The detailed Precision/Recall and F-Measure values for four "cow" related queries are shown in fig. 6.22. The x-axis of the diagrams are normalized to the size of each category, i.e. 410 images. Lines ending earlier indicate result sets smaller than the expected 410 images.

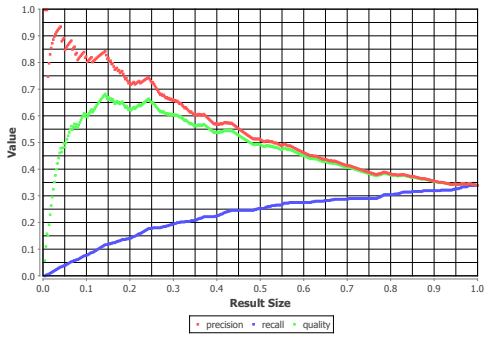
Figure 6.22(a) represents the results for the single best clause. The precision drops relatively constantly with the increasing result set size towards a value of 0.35. The maximum quality (i.e. F-Measure) has been found at a minimum similarity of 0.823. At that point, the precision is 0.84. The resulting query for the first split is:

```
W:1750@0.823
```

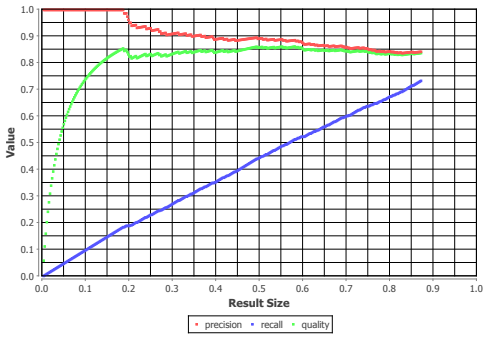


(a) (b) (c) (d) (e) (f) (g) (h)

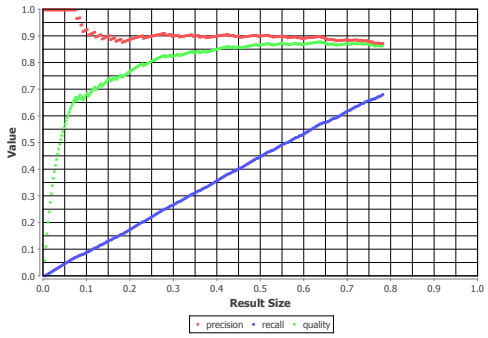
Figure 6.21.: Precision/Recall of ETH-80 categories, single feature



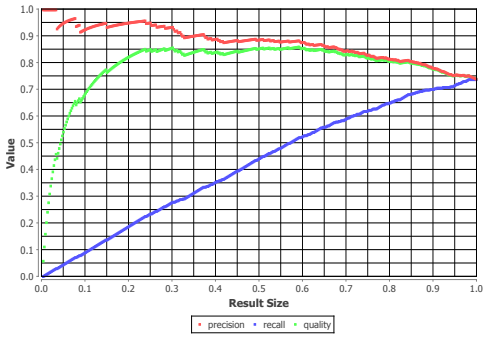
(a) Single Clause



(b) 20 SHOULD Clauses



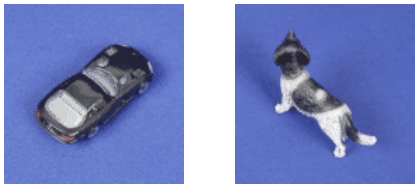
(c) 20 SHOULD Clauses with MUST_NOT



(d) Most Relevant Clauses have Tolerance

Figure 6.22.: Precision, Recall and Quality (F-Measure) for "cow" related queries

(a) rank 77 (b) rank 80 (c) rank 81 (d) rank 82 (e) rank 84



(f) rank 89 (g) rank 98

Figure 6.23.: Highest Ranked False Positives for "cow", 20 SHOULD (first 100 hits)

fig. 6.22(b) shows the results for a query containing a set of several SHOULD clauses. Until a size of 76 images, the precision achieves the optimal value of 1.0. After that threshold, it quickly drops to 0.95 and after that then the precision slowly descends. The maximum result set size with this query is 360 images, containing $tp = 304$ true positives, $fp = 56$ false positives and the not retrieved false negatives $fn = 106$. The resulting precision (eq. (3.2)) is $\pi = \frac{304}{304+56} = 0.844$ and the recall (eq. (3.3)) is $\rho = \frac{304}{304+106} = 0.741$ More images cannot be retrieved, as the query contains a minimum similarity for each clause.

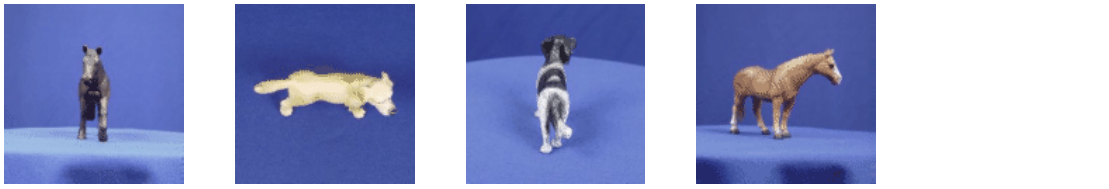
The highest ranked false positives are displayed in fig. 6.23. The first false positive is a black and white dog from different perspectives, followed by black cars with white/grey interior.

The query containing the 20 most relevant query features is:

```
W:1750@0.823  SH:2150@0.863  W:2333@0.833 W:1818@0.803
SH:1665@0.857 W:2143@0.804   W:1838@0.809 W:2083@0.773
W:2316@0.806  SH:2302@0.773  W:1921@0.814 RGB:2399@0.999
SH:1922@0.856 W:1811@0.824   W:1650@0.793 SH:1734@0.864
SH:1670@0.927 RGB:2405@0.999 W:1985@0.804 W:2151@0.799
```

Figure 6.22(c) is based on the full retrieved query containing additional negative clauses. In comparison to the previous query, the first false positive is already ranked at position 31, followed by a couple of other false positives. Again, the same dog and cars as from above are included in the results. As a first difference, the precision remains

(a) rank 35 (b) rank 37 (c) rank 41 (d) rank 48 (e) rank 55



(f) rank 63 (g) rank 68 (h) rank 74 (i) rank 94

Figure 6.24.: Highest Ranked False Positives for "cow", MUST NOT (first 100 hits)

quite stable at a level of 0.9. Out of 323 retrieved images, $tp = 286$ are true positives, $fp = 37$ are false positives and $fn = 124$ related images are missing. The resulting precision is $\pi = \frac{286}{286+37} = 0.885$ and the recall (eq. (3.3)) is $\rho = \frac{286}{286+124} = 0.698$. Thus, the precision is slightly better at the expense of the recall.

Figure 6.22(d) is based on an attempt to manually improve the calculated descriptor. The similarity thresholds for the first two positive clauses have been lowered by 10% in order to increase the result size and implicitly the recall rate.

```
(W:1750@0.741 -H:2908@0.988) (SH:2150@0.776 -H:1560@0.970)
...
```

At a result size of 410 images, the precision is $\pi = 0.74$ and the recall is $\rho = 0.74$. While the precision is lower than for the other queries above, the recall is similar to the 20 SHOULD clauses. Yet, the total possible result set is about twice as large as for the restricted ones. Thus, the full result set contains 884 retrieved images and is separately depicted in fig. 6.25. The total amount of true positives is increased to $tp = 356$ and the still not retrieved false negatives are reduced to $fn = 54$. Thus, the final precision for the complete retrieved result is $\pi = \frac{356}{356+528} = 0.403$ and the recall is $\rho = \frac{356}{356+54} = 0.868$.

The highest ranked false positives up to rank 100 are shown in fig. 6.26. The amount of false positives is reduced from 9 to 5. Especially the black car is ignored in this case.

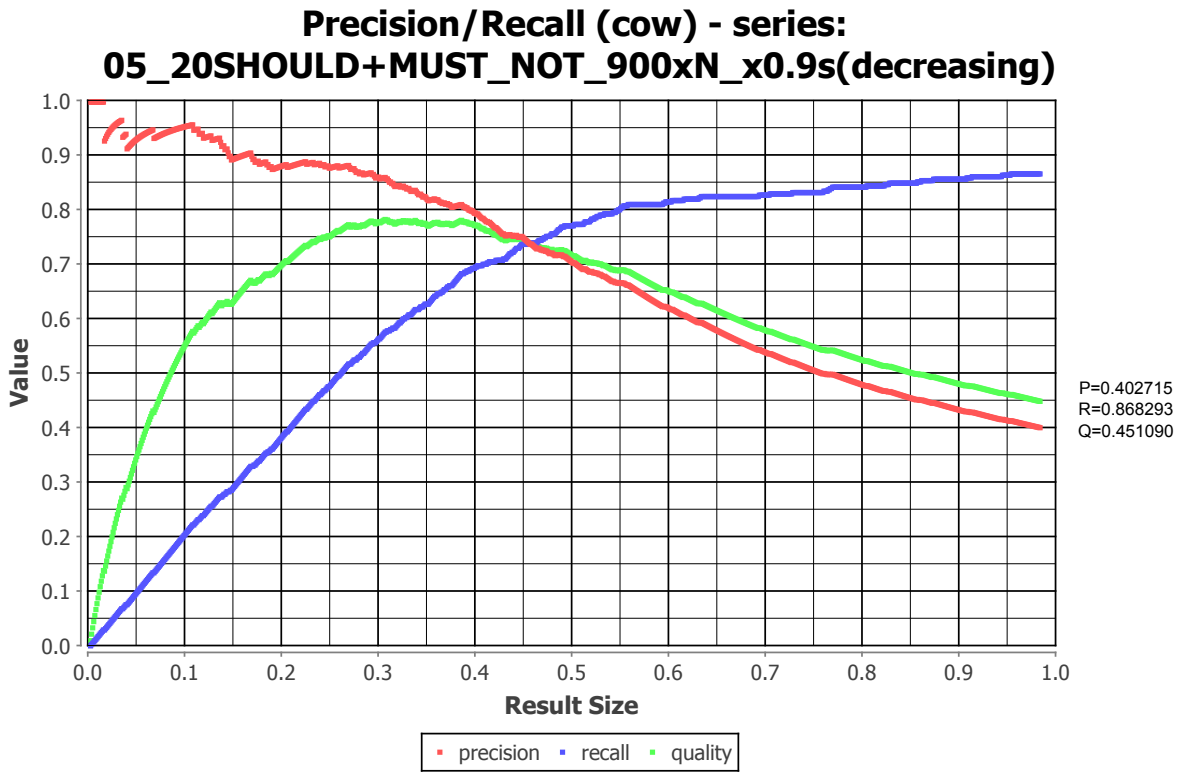


Figure 6.25.: Precision/Recall of Manually Edited Query for "cow" up to Rank 900

(a) rank 14 (b) rank 31 (c) rank 35 (d) rank 60 (e) rank 97

Figure 6.26.: Highest Ranked False Positives for "cow", modified MUST NOT (first 100 hits)

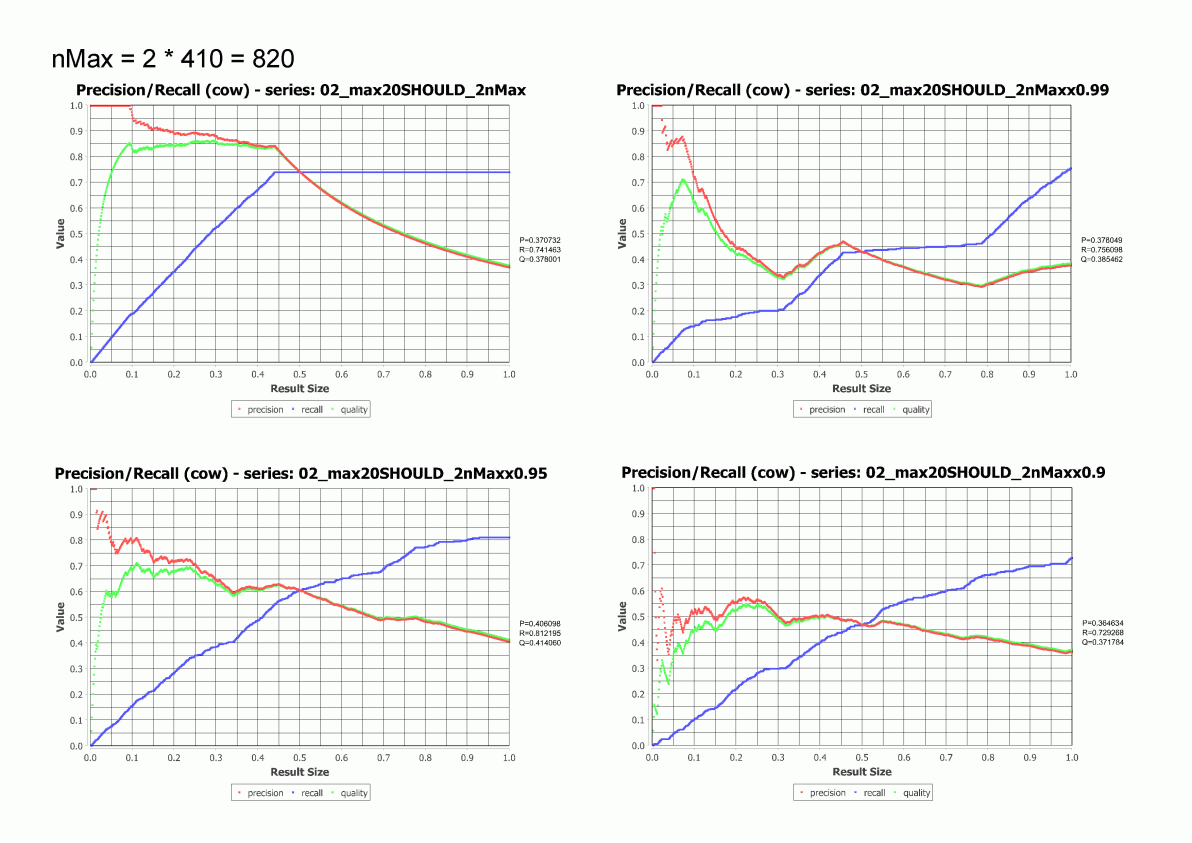


Figure 6.27.: Precision/Recall of ETH-80 "cow", max 20 SHOULD clauses, variations of "slack"

6.4.4.3. Effect of Boost Parameter

The previous section indicates limitations of the retrievable results. This is mainly caused by the strict application of the similarity threshold. It is simply cutting away further relevant images with a lower similarity to the query.

This effect can be attenuated by lowering the threshold for the most expressive query, allowing more hits to be considered. In order to enhance the query descriptor automatically, the quality of each positive top level clause is added as boost parameter. When merging multiple of those clauses, the more relevant sub results are expected to gain better ranks than the less relevant ones.

Figure 6.27 displays the impact of simply reducing the similarity threshold for all clauses. In this case, only the best 20 SHOULD clauses are used. The result size is $|r| = 2|R|$ to show the effects for a case, where the precision is not near 1.0.

In the upper left example, precision and recall of the result set are very good up to

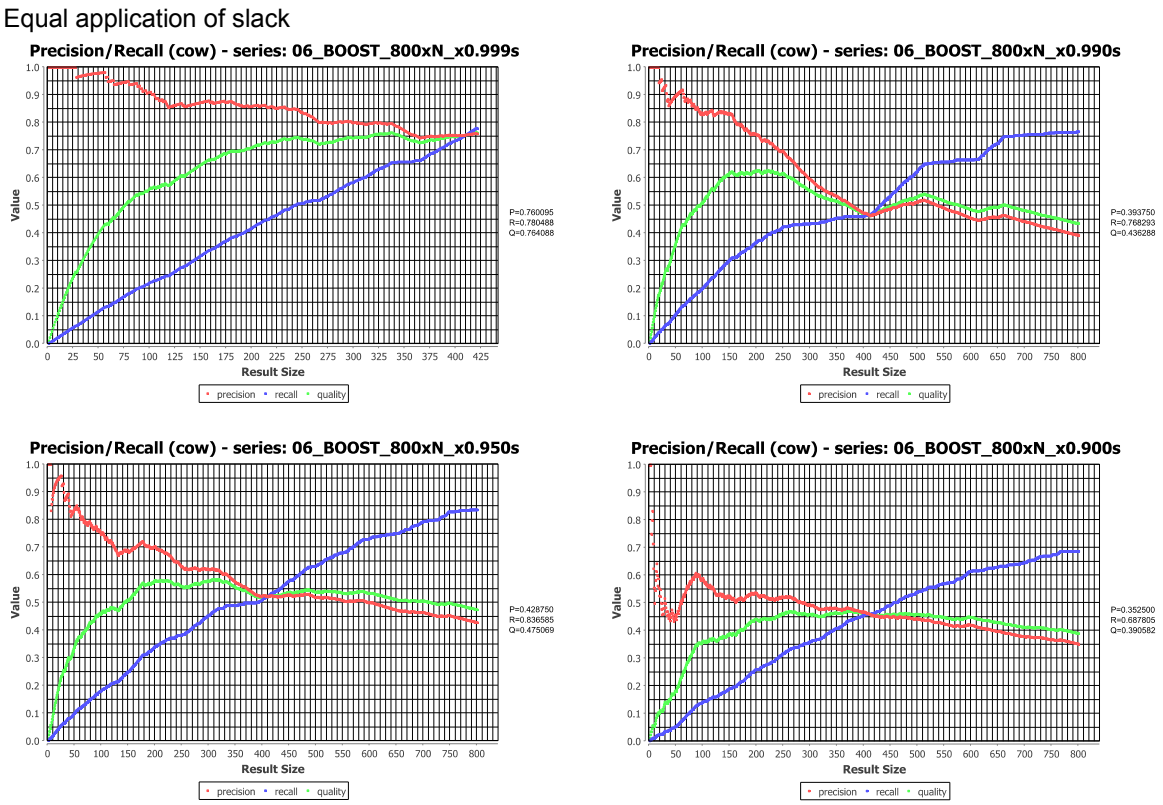


Figure 6.28.: Equal Application of Slack

the point, where the query cuts off all following results. From a result size of about 0.45, no more additional data is available. The maximum recall is 0.74

The upper right result uses similarity thresholds reduced by a factor of 0.99. The precision is much lower than in the first case. The recall increases in several steps until it reaches a slightly higer value than previously ($\rho = 0.75$).

The results of the lower left diagram are generated with thresholds reduced to 0.95. This leads to a decrease of precision between the former two tests. The recall keeps increasing until it seems to reach a limit at about 0.95 of the result size. The final value is $\rho = 0.81$.

In the lower right diagram, the thresholds are reduced by a factor of 0.9. The precision drops very quickly to a value of about 0.5 where it remains until about a size of 0.5. Then it starts decreasing further at an almost constant rate. The recall increases more slowly than before, but relatively steadily. At a result size of 1.0, it is still increasing and reaches a value of $\rho = 0.73$.

In fig. 6.28 the full queries containing also negative clauses and a boost are tested. Again, the similarity threshold of each top-level Boolean clause is reduced by different amounts.

On the upper left, the diagram shows precision and recall for a threshold reduction by 0.999. The precision slowly decreases and the recall slowly increases until it is slightly higher than the precision ($\rho = 0.78$). At 425 results, the curves stop.

The upper right diagram is based on a query with reduced thresholds of 0.99. Again, the precision drops much faster than previously below 0.5. When reaching the size of the relevant set (≈ 410), there are two more points, where the precision increases again. The final recall is 0.77.

When using a factor of 0.95 (lower left), the precision does not drop as much as before and with less obvious steps. The recall reaches a maximum of $\rho = 0.84$.

In the last case, with a factor of 0.9, the precision drops very quickly towards 0.5 from where it rises up to 0.6 and then keeps dropping steadily. The best recall possible is $\rho = 0.86$.

When adding a "slack" to the thresholds of each query, the boost is used to alter its percentage. Clauses with a boost of 1.0 would be treated with the maximum additional slack and clauses with 0.0 with none at all. In between, a linear interpolated value is used.

$$\sigma_c = 1.0 - ((1.0 - \sigma_{max}) * \beta_c) \tag{6.6}$$

where σ_c is the factor to modify a single clause c, σ_{max} is the maximum factor to be added and β_c is the boost factor assigned to clause c.

The last testing series (fig. 6.29) is the boost dependent variation of similarity thresholds. In this case, the new similarity threshold of each clause is reduced by the full amount of "slack " with a boost of 1.0 and with no modification for a boost of 0.0.

In the upper left case (factor 0.999), the results are comparable to the previous series, but with a slightly worse recall ($\rho = 0.77$).

The upper right diagram with a factor of 0.99, shows the same steps as in the former series with the equal decrease of the threshold. The final recall of 0.8 indicates an improvement.

The lower left test case plots the results for a factor of 0.95. Again, the steps are mostly gone. The recall curve starts relatively steep and then becomes more and more shallow with a short increase at the end. This case reaches a recall of 0.73.

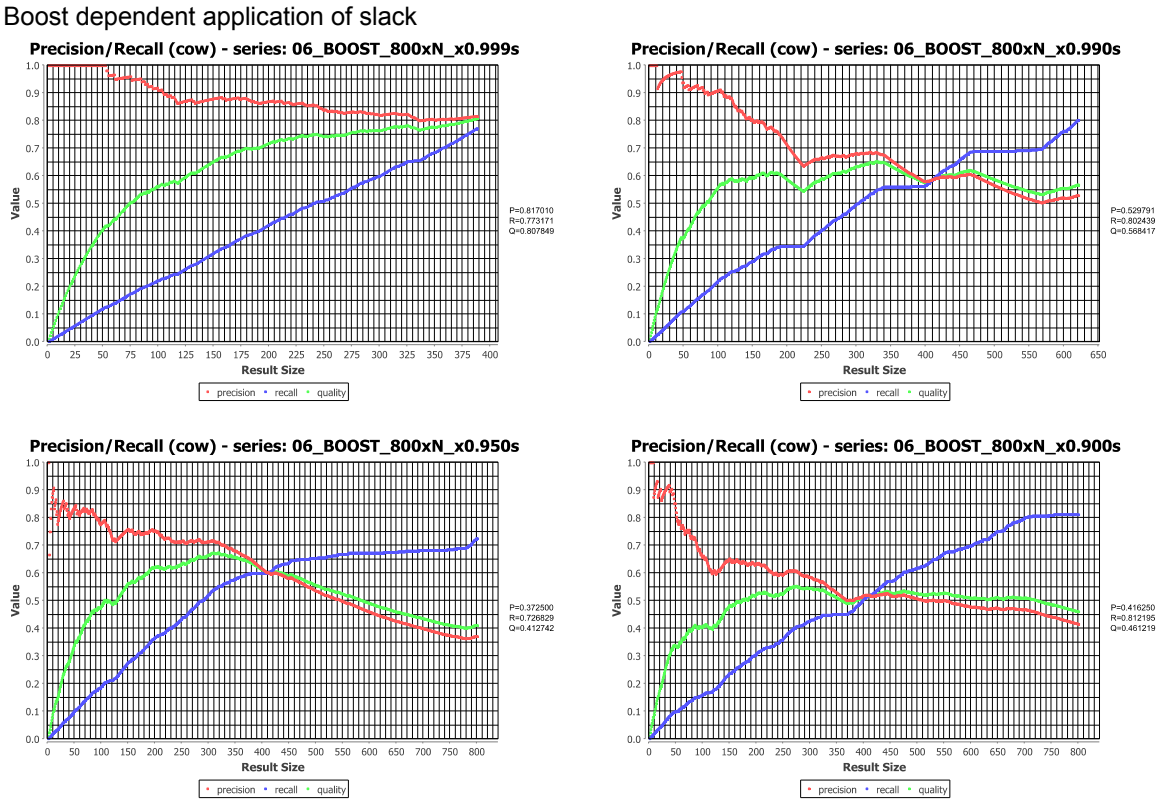


Figure 6.29.: Boost Dependent Application of Slack

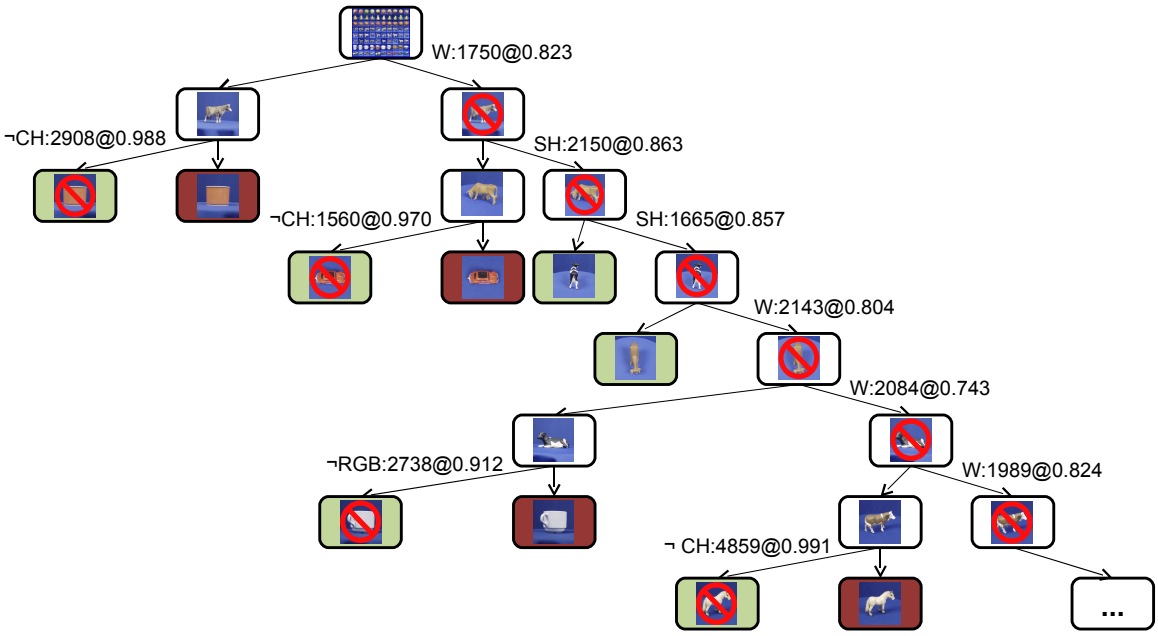


Figure 6.30.: Decision Tree for concept "cow" (most relevant clauses)
For each node split, the decision query term is specified.

In the final test with 0.9, the precision drops more quickly to 0.6 than the other curves in this series. The maximum recall is $\rho = 0.81$ with a more or less steady increase.

6.4.4.4. Constructed Category Queries

The categories "tomato", "pear" and "apple" have successfully been categorized with at most 3 to 10 positive clauses. Adding more clauses would not have had any noteworthy effect on the results, e.g. only adding a single image. The most complex queries were generated for the animals "cow", "horse" and "dog".

The full generated descriptor for the concept "cow", defined in the query language, is:

```
(W:1750@0.823 -H:2908@0.988) (SH:2150@0.863 -H:1560@0.970)
 SH:1665@0.857 W:2143@0.804   (W:2084@0.743  -RGB:2738@0.912)
(W:1989@0.824 -H:4859@0.991) SH:2302@0.773   W:1839@0.834
RGB:2399@0.999 W:1811@0.824   SH:1670@0.926   W:1985@0.799
SH:1816@0.886  W:1662@0.817   SH:1922@0.866   W:1827@0.844
W:1980@0.857   W:2151@0.799   RGB:2055@0.999  SH:1657@0.920
```

In comparison, the descriptor for the concept "tomato" only requires 3 clauses:

```
H:5793@0.988527 H:6031@0.999211 SH:6377@0.952435
```

Figure 6.30 shows the decision tree for the first half of the query for the learned concept "cow". The first split of the image set I is the query *W:1750@0.823*. It has been identified as the single query with the highest gain possible. As a result, the new left node contains a result set with an F-Measure of 0.67 and a precision of 0.84 for "cow".

As the retrieved results still contain several false positives, the query is refined by adding the negative clause *-H:2908@0.988*. This addition is filtering out a cluster of irrelevant images, i.e. a set of red cups. This second split generates a left leaf with a further improved precision and a right leaf with only irrelevant images. The node splitting is performed in the same way for the right node on the 2nd level, generating more left nodes with the highest precision possible. The right nodes contain less relevant training images for each split until the remaining training set is too small for learning or the maximum learning depth (i.e. 20 positive clauses) is reached.

6.4.5. Analysis

The extracted descriptors for each category show different characteristics (table 6.3). The categories "tomato", "pear" and "apple" can be described with short queries. The "apple" and "tomato" queries have an emphasis on simple colour based features. The samples in the ETH-80 collection have relatively uniform colours (i.e. red and green), distinct from the other categories. The "pear" has similar colours, but it's unique shape seems to be even more distinctive, whereas the descriptor is composed of the spatially aware features. Similarly, the "cup" descriptor takes advantage of the unique handle shape, but also the distinctive plain white and brown colours can be used. The "car" seems to be much more challenging, as the colours are different for each sample, but the relatively uniform shape and size clearly favourizes the spatially aware features. The most complex shapes involved are the animals. It varies depending on the point of view, and even the colours change for each single sample object. Further, the shapes of different animals are very similar from the same angle, making it even more difficult to find a suitable feature.

An obvious property of all queries is the complete lack of MUST_HAVE clauses in the results. Obviously, this is caused by the strict rule in the design section 4.3. By protecting every single true positive in combination with only a small set of feature vectors, there was no valid combination of two MUST_HAVE clauses available.

Being one of the most difficult categories in the collection (24 clauses in total table 6.3), the "cow" queries are examined in more detail (table 6.4). The precision for the best

Table 6.3.: Query Composition

Category	*SHOULD clauses*	*MUST_NOT clauses*	*RGB*	*H*	*SH*	*W*
apple	10	1	8	2	1	0
car	20	0	0	0	15	5
cow	20	4	3	3	7	11
cup	15	3	1	5	4	8
dog	20	3	0	6	11	6
horse	20	4	2	9	2	11
pear	8	0	0	0	3	5
tomato	3	0	0	2	1	0

Table 6.4.: Query Quality Parameters

	Precision	*Recall*	*F-Measure*
fig. 6.22(a)	0.341463	0.341463	0.341463
fig. 6.22(b)	0.843137	0.734146	0.838350
fig. 6.22(c)	0.875000	0.682927	0.865636
fig. 6.22(d)	0.739024	0.739024	0.739024

single clause (fig. 6.22(a)) is almost constantly dropping towards 0.35 for a result set of the size 410. This can be explained by the high diversity of relevant samples and a large number of similar images from other categories. Using 20 different clauses for retrieval (fig. 6.22(b)) results in a very high precision in the beginning. This effect is caused by the query images that usually achieve the highest possible similarity of 1.0 for at least one sub query. The precision then drops to 0.9, but the recall keeps climbing until the threshold is cutting off the less relevant images at a recall of 0.734146. A similar effect can be seen after introducing MUST_NOT clauses (fig. 6.22(c)), where the final precision is higher, but the achieved recall is lower.

To overcome the limited result set size, the thresholds for the most expressive clauses are lowered by 10%. The final recall is the highest of all tests and can still increase with larger result sets. The last query appears to be the most appropriate for CBIR applications, where several results can be displayed and quickly manually scanned. In this scenario, a relatively constantly increasing recall rate is considered more important than having an extremely high precision for only the few highest ranks.

6.4.6. Discussion

Using the existing ranking methods of the retrieval systems indicates certain limits of this learning approach. The main issue to be solved is the cut off result set at a given threshold. In many cases, the strict similarity thresholds may be sufficient in order to generate results with a high precision. The remaining relevant images can only be found by lowering the clause thresholds which unfortunately has a negative effect on the precision of the higher ranked images. An alternative ranking approach could be derived from the category descriptor described in sections 3.7.9 and 6.5. This way, not a single result is cut off and at the same time, the meaning of each single threshold of being the best trade-off, remains.

The learning algorithm is based on Decision Trees to find a suitable set of query terms, describing a concept as good as possible. As with most learning algorithms, the problems of over-fitting and unpredictable run times have to be addressed. Thus, suitable heuristics for this learning scenario need to be developed.

6.4.7. Summary

The DT based learning approach investigated above is capable of determining a set of suitable QBE clauses. It automatically chooses the best describing feature vector in each case and combines them into a single human-readable query. The main challenges are to achieve a better scalability and to find a reasonable trade-off between query size and over fitting.

6.5. Query Descriptors

The category-related queries learned in section 6.4 can be used for categorization tasks. Yet, the CBIR ranking algorithm is unsuitable for direct use, as it would only return similarities above 0.0 for images within the similarity tolerance specified for the clauses. Unknown images with a low similarity could not be classified at all. Thus, a category probability must be defined even for those cases section 3.7.9.

Query descriptors derived from the prototype suffer from the problem of using highly different feature vectors. Dependent on the feature used in classification, the extracted raw probability from a single positive term may vary largely. Thus, it is required to adjust the similarity curves for each feature type to a similar behaviour (see section 6.1).

The testing has been carried out on a Solaris 10 x86 server system with 4 CPU-cores used in parallel.

6.5.1. Requirements

- Image set of annotated categories
- Normalized feature vectors
- Set of descriptors, one for each category

6.5.2. Testing

The test is performed on the ETH-80 image set, containing 8 equally sized categories. In order to categorize a single image, the normalized probabilities for this image of belonging to each category are calculated (eq. (3.31)), resulting in a list of values between 0.0 and 1.0.

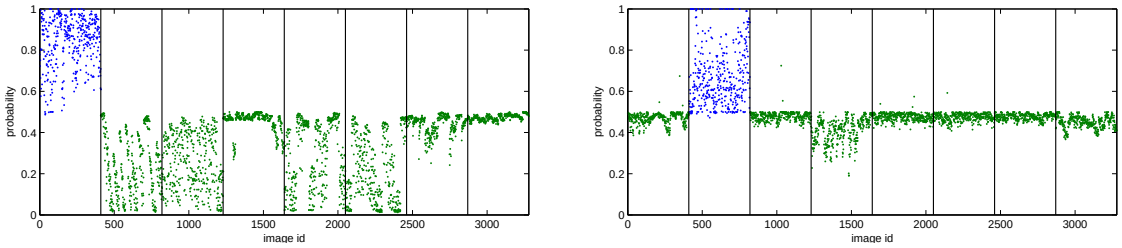
At the first stage, these raw probabilities are used for a first descriptor-wise categorization. To reduce the amount of ambiguous categorizations, the range $\phi = [0.45, 0.55]$ is considered as being "unsure". Probabilities above 0.55 are interpreted as positive classification and values below 0.45 are negatives. At the second stage, all probabilities $\Phi(i)$ for a single image are merged. Out of these probabilities the highest one is selected eq. (3.34) and the "unknown" image is categorized this way. In both cases, the categorizations are compared to the real category of this image and the correctness rate for each category is calculated.

6.5.3. Results

The scatter plots in fig. 6.31 visualizes all probabilities for the learned descriptors. The horizontal axis stands for the respective image id. These ids are sorted by category and every section contains the 410 images of the same category[1]. The relevant images for the respective category are highlighted by a different colour.

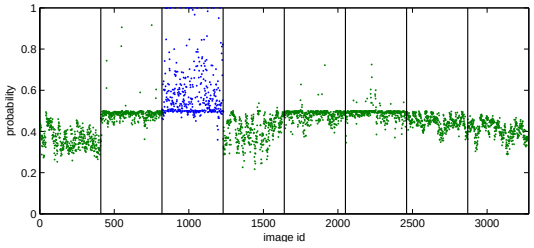
Generally, the plots in fig. 6.31 show similar characteristics. Usually, the true positives achieve probability values above the normalized threshold of 0.5. In some cases, false positives are clearly below. The negatives show similar patterns:

[1]Due to parallel threads during the image import, image 410 is a "car" and 411 is an "apple". In the diagram, the wrong horizontal locations are barely noticeable and the colours are correct. In the summary, the category names were used instead of the ids.

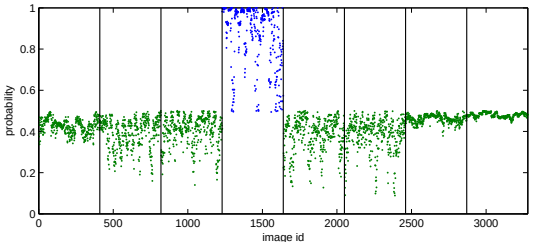


(a) apple

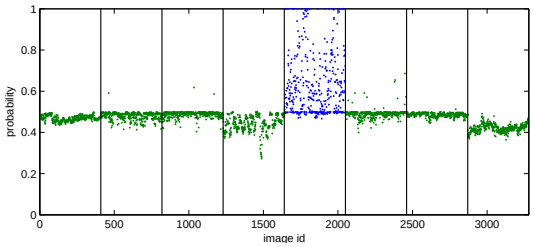
(b) car



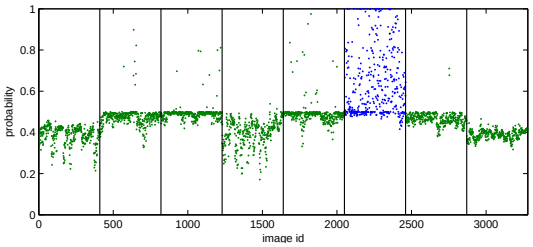
(c) cow



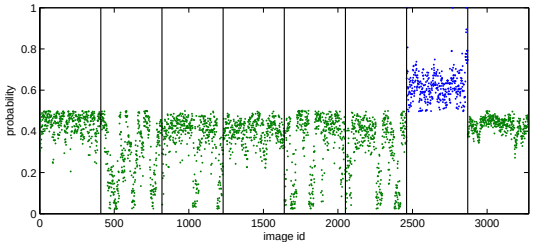
(d) cup



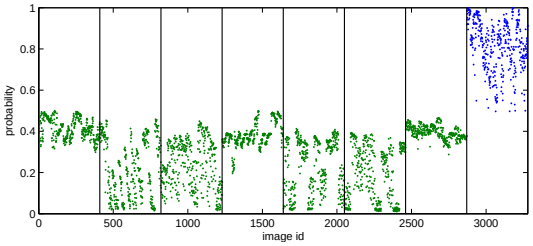
(e) dog



(f) horse



(g) pear



(h) tomato

Figure 6.31.: Category Probabilities for ETH-80 Images

(a) The probabilities are aggregated very close to 0.5. This is the only case, where false positives appear.

(b) The probabilities are mostly lower than 0.5 but still aggregated.

(c) The probabilities are spread across the whole range $[0.0, 0.5]$.

Figure 6.31(a) is generated with the "apple" descriptor. The apple related images achieve a ϕ value in the upper half of the scale, in most cases clearly above the threshold of 0.5. None of the other images is located in the upper half. The images of four categories ("car", "cow", "dog" and "horse") are relatively evenly spread across the range $[0.0, 0.5]$. The three remaining categories are mostly close to the central region around 0.5.

In fig. 6.31(b), the "car" related images are largely located in the upper half with a certain amount close to the threshold. Except from the "cup", which achieves several probabilities down to 0.2, most irrelevant images are located near 0.5. A few images from "apple", "cow", "dog" and "horse" achieved a ϕ clearly above 0.5.

The "cow" descriptor (fig. 6.31(c)) calculated about half of the relevant images close to 0.5 and quite a few irrelevant images from "car", "dog" and "horse" achieved a value above 0.5. The two results from "dog" (fig. 6.31(e)) and "horse" (fig. 6.31(f)) are similar.

For the category "cup", fig. 6.31(d) indicates neither clear false negatives nor false positives. The categories "apple", "pear" and "tomato" are located close to the 0.5, while the remaining ones may drop down below $\phi = 0.1$.

The last two categories "pear" (fig. 6.31(g)) and "tomato" (fig. 6.31(h)) again share similar profiles with no false positives. Most "pear" probabilities are between 0.5 and 0.75 and "tomato", "apple" and "cup" are the categories with many values near 0.5. The "tomato" descriptor achieves more true positive probabilities up to 1.0. Again, the other fruits and "cup" are scattered around the central area.

6.5.4. Analysis

To provide an overview of the probability values achieved by the descriptors, table 6.5 summarizes the raw probabilities for each class and descriptor. The results for every descriptor are summarized in three rows. The eight category columns represent the sets of 410 images of each category. Each box contains the amount of raw positives (ϕ >0.55), negatives (ϕ <0.45) and unclear ($\phi \in [0.45, 0.55]$) summed up to 100%. The bold numbers highlight the true positives. These values do not consider the other

Table 6.5.: Raw Categorization Summary

		(%) by real category								
descriptor	ϕ	apple	car	cow	cup	dog	horse	pear	tomato	total
apple	>0.55	**97.07**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	12.13
	[0.45, 0.55]	2.93	12.20	4.88	79.76	26.34	10.24	57.80	95.12	36.16
	<0.45	0.00	87.80	95.12	20.24	73.66	89.76	42.20	4.88	51.71
car	>0.55	0.24	**77.56**	0.49	0.00	0.24	0.24	0.00	0.00	9.85
	[0.45, 0.55]	81.71	22.44	90.49	28.05	90.24	89.27	94.15	51.46	68.48
	<0.45	18.05	0.00	9.02	71.95	9.51	10.49	5.85	48.54	21.68
cow	>0.55	0.00	1.95	**45.61**	0.00	1.22	2.68	0.00	0.00	6.43
	[0.45, 0.55]	4.63	90.73	53.66	31.22	85.85	90.00	50.73	14.15	52.62
	<0.45	95.37	7.32	0.73	68.78	12.93	7.32	49.27	85.85	40.95
cup	>0.55	0.00	0.00	0.00	**93.66**	0.00	0.00	0.00	0.00	11.71
	[0.45, 0.55]	25.85	22.44	32.44	6.34	24.63	17.32	75.12	99.27	37.93
	<0.45	74.15	77.56	67.56	0.00	75.37	82.68	24.88	0.73	50.37
dog	>0.55	0.00	0.24	0.49	0.00	**60.49**	2.20	0.00	0.00	7.93
	[0.45, 0.55]	90.73	90.73	97.32	45.85	39.51	94.88	99.27	6.83	70.64
	<0.45	9.27	9.02	2.20	54.15	0.00	2.93	0.73	93.17	21.43
horse	>0.55	0.00	1.71	2.20	0.00	3.66	**53.66**	0.49	0.00	7.71
	[0.45, 0.55]	1.22	80.98	93.90	19.27	86.10	44.88	76.59	0.24	50.40
	<0.45	98.78	17.32	3.90	80.73	10.24	1.46	22.93	99.76	41.89
pear	>0.55	0.00	0.00	0.00	0.00	0.00	0.00	**85.61**	0.00	10.70
	[0.45, 0.55]	49.51	14.88	18.05	17.32	29.02	14.88	14.39	31.22	23.66
	<0.45	50.49	85.12	81.95	82.68	70.98	85.12	0.00	68.78	65.64
tomato	>0.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**96.83**	12.10
	[0.45, 0.55]	29.76	0.98	0.00	18.29	0.00	0.00	3.41	3.17	6.95
	<0.45	70.24	99.02	100.00	81.71	100.00	100.00	96.59	0.00	80.95

descriptors. The table shows the discriminative power of each single descriptor on its own.

The descriptors for "apple" (97.07%), "cup" (93.66%) and "tomato" (69.83%) and "pear" (85.61%) achieve a recognition rate of over 85% without any false positive. The remaining positive images are all in the "unsure" area, close to $\phi = 0.5$. All the images from other categories are either located in the "unsure" or the negative area.

The "car" descriptor is still in a relatively convenient area. With a rate of 77.56% of correctly categorized positives, this descriptor is quite good. A few false positives are contained ("apple", "cow", "dog" and "horse"), but the total rate is only about 1.25%. Again, all of the remaining positives are categorized as "unsure".

The animal descriptors achieve the worst results. The recognition rate for "cow" (45.61%), "dog"(60.49%) and "horse" (53.66%) are the lowest ones in the test. Also the amount of false positives reaches up to $\approx 8\%5$. Also, the categories "cow" and "horse" are the only ones where relevant images were categorized as irrelevant.

The amount of correct categorizations is summarized in table 6.6. The decision for a single category is defined in eq. (3.34), which is determined by the highest probability for each image to be categorized. The recognition rate by the combined approach is higher than for the raw values, because no area of uncertainty has been defined. Because of merging the results of all eight descriptors, it is much easier to deal with probabilities

Table 6.6.: Categorization Summary

category	apple	car	cow	cup	dog	horse	pear	tomato	total
size	410	410	410	410	410	410	410	410	3280
correct	408	393	351	409	346	316	406	410	3024
rate (%)	99.51	95.85	85.61	99.76	84.39	77.07	99.02	100.00	92.65

close to 0.5. Even if the probability of a relevant image is below 0.5, it can still be categorized correctly, if the other descriptors return an even lower value.

6.5.5. Discussion

The testing results indicate that the queries generated by the learning algorithm 3.7.7 do in fact have sufficient discriminative power to be used as category descriptors. The repository specific normalization of the feature vectors (section 3.4.2), the conversion of similarities to probabilities (section 3.7.9) and choosing the maximum category seem to be sufficient to achieve reasonable recognition rates. In CBIR scenarios, these descriptors could also be applied to automatically assign a small set of labels to unknown images.

The probability calculations within the descriptor may also be an alternative to the current ranking algorithm which is cutting of all results below a certain similarity. The new approach takes this similarity threshold into account and also allows for expanding the results to any size. This behaviour would overcome the problems about the altered thresholds, discussed in section 6.4.

A conspicuous amount of probabilities are very close to the threshold of 0.5. This behaviour is linked to be amount of clauses in the query and the normalization based on the clause thresholds. The more clauses are in a descriptor, the more irrelevant images are close to the determined thresholds. Equation (3.31) needs to be refined further in the future and also make use of the boost parameter.

Some descriptors still struggle to capture a category while others work well. This is likely to be caused by two factors. First, the query descriptors have a strictly limited size. Having a total of only 20 positive examples and a maximum of 60 (less important) negative examples restricts the system to capture at most 20 distinct clusters in the optimal case. Allowing for more clauses may improve the recognition rates, but this also bears the risk of over fitting. The second factor to be considered is the small set of relatively simple feature vectors provided. The system could only analyze the colours and histograms as well as a single wavelet approach to recognize textures. All these features are global and do not take real advantage of the additional shape information provided.

The only two features using the provided pixel maps are the *RGB Mean* (3 dimensions) and the *Histogram* (12 dimensions) to ignore the blue background area. The use of additional, more sophisticated feature vectors is expected to boost the performance of the system.

6.5.6. Summary

Using the retrieval queries from the supervised learning for shows a high recognition rate (92.65%). This expressiveness has been achieved by using only four global feature vectors. It is expected, that this accuracy level within a CBIR system is more than sufficient.

6.6. Semi-Supervised Learning

This section deals with semi-supervised learning. It compares the results of the previous section 6.4 to learning process with a reduced training set. A leave-one-object-out cross validation on the ETH-80 dataset has been published by Leibe and Schiele [68]. This case study compares the outcome of the semi-supervised learning within this thesis to the reference approach. Their categorization attempt uses seven different methods:

1. Colour – 3 colour channel histograms [129]
2. Texture – first derivates in x/y direction [115]
3. Texture – gradient magnitude and Laplacian [115]
4. Global Shape – single, global eigenspace for all categories [82]
5. Global Shape – class specific eigenspaces [135]
6. Local Shape – "dynamic programming" contour matching [10]
7. Local Shape – greedy one-to-one contour matching Leibe and Schiele [68]

The testing has been carried out on a Solaris 10 x86 server system with 4 CPU-cores used in parallel.

6.6.1. Requirements

- Image set of annotated categories
- Annotated set of objects for each category
- Normalized feature vectors to calculate probability for an item to belong to a given descriptor

6.6.2. Testing

Data set sized for the leave-one-object-out cross validation for each single object:

- Evaluation set: 41
- Training set: $41 * 9 = 369$
- Negative set: $410 * 7 = 2870$

Evaluation steps:

1. Learn a descriptor for 9 out of 10 objects
2. calculate probability of 10th object of belonging to descriptor
3. compare probabilities for all classes/descriptors

The learning algorithm is run for each single object from the image set. The evaluation set E is the collection of 41 images for the single object to be left out. The training set T consists of the remaining $41 * 9 = 369$ images of the same category. As negative set N, the $410 * 7 = 2870$ images from the other 7 categories are used. This results in a set of 80 descriptors, each one based on learning $\frac{9}{10}$ of a single category.

As in section 6.5, each image is fed into 8 descriptors, one for each category. The category with the best calculated probability is selected. The difference to the descriptors from the supervised learning is the use of the new descriptors where the object to be identified is missing from the training set.

The reference results by Leibe and Schiele [68] would require the system to learn for each single object to learn a full set of descriptors because of the negative set. Because of the large overhead, only the 80 descriptors for the modified categories to be learnt are calculated. The remaining 7 descriptors are replaced from the pool of the 8 generic

descriptors. To allow for a better comparison, two separate test runs are performed. In the first one, the existing descriptors are used and in the second one, only the positive clauses from the existing descriptors are applied. This simulated loss of accuracy is expected to be larger than the benefit from having the negative sets with all 10 objects during learning.

6.6.3. Results

In general, the scatter plots of the 80 object specific runs are comparable to those in section 6.5.3. To provide a flavour of the differences, fig. 6.32 shows all 10 plots for the category "cow". The amount of false positives changes form object to object, especially in the already error-prone categories "car", "dog" and "horse". Another change is the amount of false negatives. The plots in figs. 6.32(c) and 6.32(i) reveal quite a few relevant images with low probabilities. Nevertheless, most of the other plots show a smaller amount of those low values than in the fully supervised learning approach (fig. 6.31(c)).

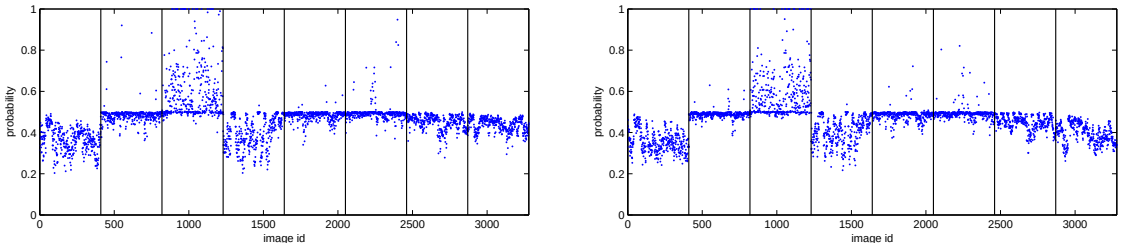
6.6.3.1. Impact of removed MUST_NOT

In a second test run, all "MUST_NOT" clauses were removed from the 80 descriptors and the new categorization results are compared to the previous ones. In addition to the originally wrong assigned categories, 6 additional images were mis-categorized. The total amount of traceable changes were 33 cases.

All critical changes are listed in table 6.7. The table compares the original descriptor to the modified one. The first column contains the image id as well as the name of the unknown object. The second one indicates whether the following results are based on the original of the modified descriptor. The eight subsequent columns contain the probabilities for each one of the descriptors used for categorization. The last column lists the categorization result. The bold numbers highlight the changes in probability leading to the categorization error.

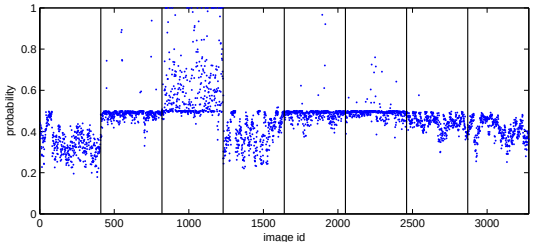
In four out of the six cases, the original probability for the correct class lies above 0.5. In the other two cases, it is slightly below but still above 0.49, indicating that the image is just below the threshold of the descriptor. All increased probabilities for the wrong category are above 0.5.

In 12 further cases, the result category also changed. Yet, these changes do not appear in the final result as the original results were also wrong in the first place. Finally, in 15 cases, the numerical probabilities changed, but these differences are too small to have

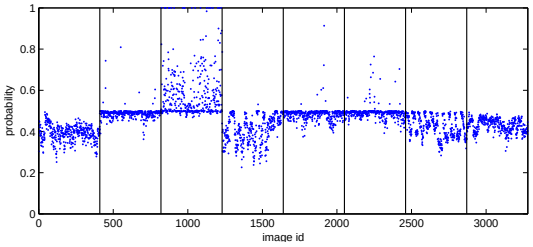


(a) cow1

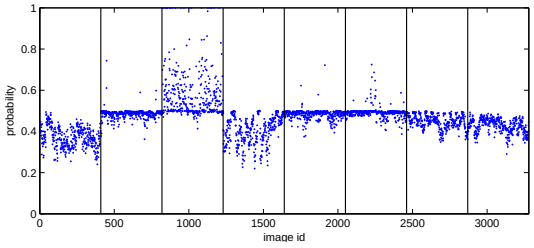
(b) cow2



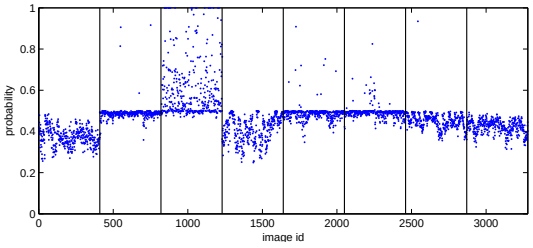
(c) cow3



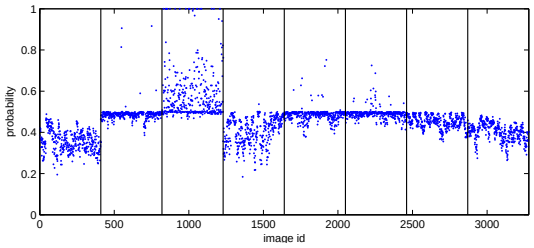
(d) cow4



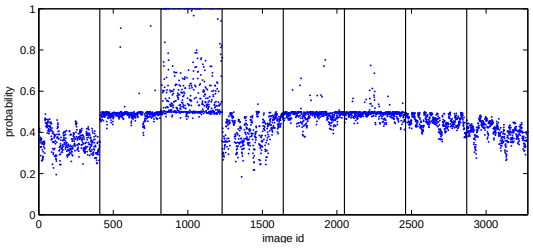
(e) cow5



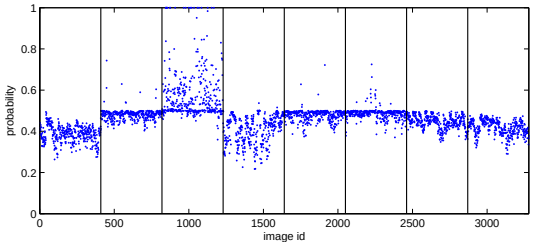
(f) cow6



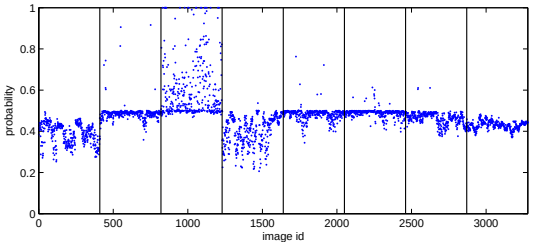
(g) cow7



(h) cow8



(i) cow9



(j) cow10

Figure 6.32.: Category Probabilities for ETH-80 Images (all cows)

Table 6.7.: Additional False Categorizations

object	series	apple	car	cow	cup	dog	horse	pear	tomato	result
776	original	0.014	0.670	0.485	0.367	0.477	0.475	0.109	0.015	car
car7	modified	0.014	**0.670**	0.485	0.367	0.477	**0.848**	0.109	0.015	horse
813	original	0.229	0.494	0.480	0.468	0.480	0.476	0.454	0.355	car
car9	modified	0.229	**0.494**	**0.510**	0.468	0.480	0.476	0.454	0.355	cow
1104	original	0.050	0.489	0.498	0.487	0.481	0.496	0.436	0.077	cow
cow6	modified	0.050	0.489	**0.498**	0.487	**0.545**	0.496	0.436	0.077	dog
2114	original	0.473	0.443	0.468	0.429	0.590	0.611	0.417	0.385	horse
horse10	modified	0.473	0.443	0.468	0.429	**0.662**	**0.611**	0.417	0.385	dog
2248	original	0.293	0.483	0.493	0.422	0.487	0.498	0.417	0.354	horse
horse4	modified	0.293	0.483	**0.606**	0.422	0.487	**0.498**	0.417	0.354	cow
2348	original	0.248	0.490	0.498	0.396	0.495	0.577	0.437	0.319	horse
horse7	modified	0.248	0.490	0.498	0.396	**0.638**	**0.577**	0.437	0.319	dog

Table 6.8.: Recognition Results for the categorization of unknown objects [68]

	Color	D_xD_y	Mag-Lap	PCA Masks	PCA Gray	Cont. Greedy	Cont. DynProg	Avg.
apple	57.56%	**85.37%**	80.24%	78.78%	**88.29%**	77.07%	76.34%	77.66%
pear	66.10%	90.00%	85.37%	**99.51%**	**99.76%**	90.73%	91.71%	89.03%
tomato	**98.54%**	94.63%	**97.07%**	67.80%	76.59%	70.73%	70.24%	82.23%
cow	86.59%	82.68%	**94.39%**	75.12%	62.44%	86.83%	86.34%	82.06%
dog	34.63%	62.44%	74.39%	72.20%	66.34%	**81.95%**	**82.93%**	67.84%
horse	32.68%	58.78%	70.98%	77.80%	77.32%	**84.63%**	**84.63%**	69.55%
cup	79.76%	66.10%	77.80%	**96.10%**	**96.10%**	**99.76%**	**99.02%**	87.81%
car	62.93%	**98.29%**	77.56%	**100.0%**	**97.07%**	**99.51%**	**100.0%**	90.77%
total	64.85%	79.79%	82.23%	83.41%	82.99%	86.40%	86.40%	80.87%

any effect on the result.

6.6.3.2. Reference System

Table 6.8 summarizes the results of the reference system by Leibe and Schiele [68] to allow for a direct comparison with the approach proposed in this thesis. These values are used in the analysis section below.

6.6.4. Analysis

Table 6.9 is constructed in the same way as described in section 6.5.4, giving an impression of how good a single descriptor is able to categorize the unknown images.

In the cross validation, the amount of false negatives is increased in comparison to the fully supervised learning (table 6.5). Only the categories “car”, “pear” and “tomato”

Table 6.9.: Raw Categorization Summary

		(%) by real category								
descriptor	ϕ	apple	car	cow	cup	dog	horse	pear	tomato	total
apple	>0.55	**77.32**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.66
	[0.45, 0.55]	20.73	12.20	4.88	79.76	26.34	10.24	57.80	95.12	38.38
	<0.45	1.95	87.80	95.12	20.24	73.66	89.76	42.20	4.88	51.95
car	>0.55	0.24	**55.85**	0.49	0.00	0.24	0.24	0.00	0.00	7.13
	[0.45, 0.55]	81.71	44.15	90.49	28.05	90.24	89.27	94.15	51.46	71.19
	<0.45	18.05	0.00	9.02	71.95	9.51	10.49	5.85	48.54	21.68
cow	>0.55	0.00	1.95	**24.88**	0.00	1.22	2.68	0.00	0.00	3.84
	[0.45, 0.55]	4.63	90.73	73.90	31.22	85.85	90.00	50.73	14.15	55.15
	<0.45	95.37	7.32	1.22	68.78	12.93	7.32	49.27	85.85	41.01
cup	>0.55	0.00	0.00	0.00	**71.22**	0.00	0.00	0.00	0.00	8.90
	[0.45, 0.55]	25.85	22.44	32.44	24.88	24.63	17.32	75.12	99.27	40.24
	<0.45	74.15	77.56	67.56	3.90	75.37	82.68	24.88	0.73	50.85
dog	>0.55	0.00	0.24	0.49	0.00	**32.93**	2.20	0.00	0.00	4.48
	[0.45, 0.55]	90.73	90.73	97.32	45.85	66.59	94.88	99.27	6.83	74.02
	<0.45	9.27	9.02	2.20	54.15	0.49	2.93	0.73	93.17	21.49
horse	>0.55	0.00	1.71	2.20	0.00	3.66	**20.24**	0.49	0.00	3.54
	[0.45, 0.55]	1.22	80.98	93.90	19.27	86.10	79.02	76.59	0.24	54.66
	<0.45	98.78	17.32	3.90	80.73	10.24	0.73	22.93	99.76	41.80
pear	>0.55	0.00	0.00	0.00	0.00	0.00	0.00	**83.17**	0.00	10.40
	[0.45, 0.55]	49.51	14.88	18.05	17.32	29.02	14.88	16.83	31.22	23.96
	<0.45	50.49	85.12	81.95	82.68	70.98	85.12	0.00	68.78	65.64
tomato	>0.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**94.88**	11.86
	[0.45, 0.55]	29.76	0.98	0.00	18.29	0.00	0.00	3.41	5.12	7.20
	<0.45	70.24	99.02	100.00	81.71	100.00	100.00	96.59	0.00	80.95

Table 6.10.: Categorization Summary (Leave-One-Object-Out)

category	apple	car	cow	cup	dog	horse	pear	tomato	total
size	410	410	410	410	410	410	410	410	3280
	results								
correct	366	353	279	340	270	211	404	410	2639
rate (%)	89.27	86.10	68.05	82.93	65.85	51.46	98.54	100.00	80.27
	reference (Leibe and Schiele [68])								
avg.(%)	77.66	90.77	82.06	87.81	67.84	69.55	89.03	82.23	80.87
best (%)	88.29	100.00	94.39	99.76	82.93	84.63	99.76	98.54	93.54

are not missing any relevant images. The highest rate for false positives is the "horse" descriptor dealing with "dog" images (3.66%).

The "tomato" loses about 5% and the pear 2.5% of their previous recognition rate. A clear decrease for the "apple" and "cup" descriptors can be observed, dropping from 97.07% to 77.32%, and from 93.66% to 71.22% respectively. Most of the missing ones have been moved to the area of uncertainty around the threshold. For the "cup", a large set of 3.9% relevant images are considered to be irrelevant.

Amongst the more difficult categories, the rate for the car dropped to 55.85% but all remaining ones are located in the area of uncertainty. The remaining animal categories share the lowest positive rates, ranging from 32.93% ("dog") down to 20.24% ("horse").

Table 6.10 provides an overview of the final categorization results compared to the reference by Leibe and Schiele [68]. The table shows the category-wise recognition rates as well as both the average and best results from the reference.

In two cases, the proposed learning approach slightly outperforms the best results of the reference ("apple", "tomato") and in one case, the results are close together ("pear"). For all other classes, is more or less below the average recognition rate. In total, the results are close to the average, both slightly above 80%. In case of the modified queries with the negative clauses missing, the total result would be $\frac{6}{3280}100\% \approx 0.182\%$ lower than the current value.

Table 6.11 and fig. 6.33 analyze in detail, which particular objects caused most errors. These objects are collected in fig. 6.34.

The "apple" category was misclassified 44 times. From these errors, 30 were caused by a single object, namely "apple10" (fig. 6.34(a)).

The errors for each "car" are distributed between 1 and 11. "car1" (fig. 6.34(b)) is the one causing most errors, but there is no highly specific problem with this one. In general, the cars have much variation in colours and the features used are all related to

Table 6.11.: Wrong Classification by Object

category	objects										total
apple	3	30	0	7	0	0	4	0	0	0	44
car	11	1	2	3	6	7	9	10	1	7	55
cow	10	10	6	9	11	31	16	7	16	15	131
cup	0	19	0	0	0	17	13	1	9	11	70
dog	14	11	17	19	14	18	26	6	9	6	140
horse	16	22	21	25	31	13	23	14	11	23	199
pear	0	0	3	0	0	0	0	2	0	1	6
tomato	0	0	0	0	0	0	0	0	0	0	0

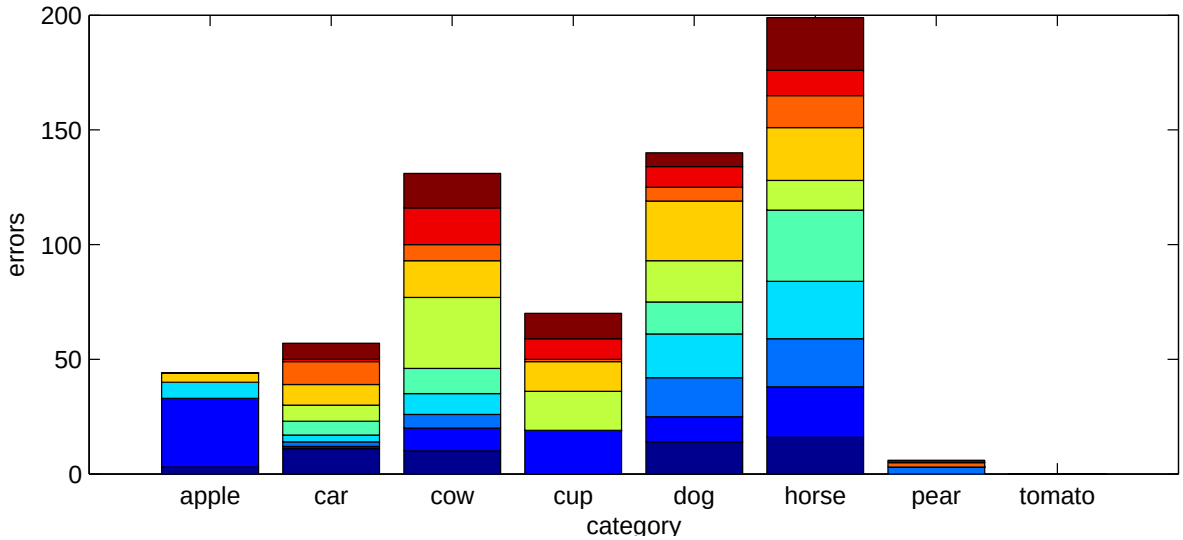


Figure 6.33.: Wrong Classification by Object

(a) apple10 (b) car1 (c) cow5 (d) cup10 (e) dog6 (f) horse4

Figure 6.34.: Most Difficult Objects

colours, making it hard to find suitable sample images.

For the "cow", the most difficult object is "cow5" (fig. 6.34(c)) with 31 relevant images missing, probably caused by two reasons. First, other than the other ones, this cow is lying on the ground and thus has a different shape. Also, the black-and-white pattern is similar to a dog's pattern.

In the "cup" category, the object "cup10" (fig. 6.34(d)) was missed 19 times. This is probably due to the specific shape and unique gray/green color. The side views of this cup only achieved a probability below 0.4. In 14 of these cases, the object has bee mistaken as "cow".

Similar to the "cow" category, the "dog" contains multiple difficult objects. The worst one with 26 errors is "dog6" (fig. 6.34(e)), with a black and white pattern. Not surprisingly, it has been mistaken as a cow 15 times.

The most difficult category is the "horse". Every single object has at least 11 errors out of 41 samples. The highest error of 31 is caused by "horse4" fig. 6.34(f). Again, the high variation of colour patterns and a missing shape/contour feature seem to be responsible for these results.

The last two categories "pear" and "tomato" performed best. While the "pear" only caused 6 false results, the "tomato" has been recognized correctly always. In both cases, the highly homogeneous colours and the simple shapes could be captured by the system.

6.6.5. Discussion

In the direct comparison with the reference system, the results are very similar. Thus, the proposed learning algorithm generating a low-level query can compete with the learning approach used in the reference. This outcome is satisfying, as the main purpose of the learning algorithm was to provide evidence for the capabilities of the query language in general.

The leave-one-object-out cross validation suffers from the same weaknesses as the supervised test (section 6.4). The feature vectors currently available are simply not capable to distinguish between the complex "animal" categories. In the reference system, these categories as well as "car" and "cup" achieved high recognition rates with the "Contour" features that were not available for this test.

Further, only having a set of existing images available for querying, restricts the possibilities of learning. Modern machine learning usually heavily relies on complex structures such as neural networks or support vector machines to learn a given concept. These approaches allow a fine grained approximation of the desired feature without the

need of finding suitable query images from the existing training set. On the other hand, they usually require defined types of feature vectors and a reasonable understanding of the particular meaning.

6.6.6. Summary

The semi-supervised learning outcome cannot compete with recent SVM approaches (e.g. [117]). Yet, it is comparable to earlier studies that are several more complex features, such as shape information Leibe and Schiele [68].

It is expected that adding new feature vector plug-ins to the proposed system makes it much more competitive without modifying the underlying theory and the need of feature specific optimizations. At the same time, the queries used within the categorization remain understandable by experienced users. This allows for subsequent and target-oriented modifications of an existing categorization query.

6.7. Discussion

Before tackling the research question itself, several preconditions had to be met. Each case study described above investigates one essential aspect of CBIR. Building up on top of each other, the case studies finally allow for an analysis of how good the query language can map a set of given low-level features to a higher semantic concept. Every query descriptor learnt by the system represents a possible semantic mapping for the related category.

In section 6.1, it is examined, how each feature vectors reacts to changes of QBE images. A reduction of the image size has an impact on the result whereas the robustness of each single feature vector varies.

The extracted normalization functions in section 6.2 are found to be relatively accurate for the related repository, but may be unsuitable for others. The main advantage of this normalization is the better comparability of distinct feature vectors, allowing for a more accurate fusion, which is necessary for the learning and classification tasks.

It is shown in section 6.3, that some categories are relatively easy to be learnt (e.g. the grayscale cars), while others cannot be captured (e.g. the mostly random background images) with the feature vector set used. Further, particular strengths and weaknesses of each feature vector can be pointed out before the actual learning process.

The supervised machine learning in section 6.4 works well, as long as the feature

vectors have a sufficient discriminative power, as mentioned in section 6.3. In some cases (i. e. the animals), the feature vectors used prove to be not capable of capturing highly discriminative features for a high result quality. This still seems to be sufficient for most retrieval tasks. The first 410 results can easily reach a precision of $\pi > 0.5$ and in several set-ups even $\pi > 0.8$. This can be tolerated, as users could easily filter out the false positives and only persevering users are expected to keep searching beyond this amount of images.

Section 6.5 shows that the categorization is more challenging than a retrieval, where the user itself is part of the final "filter". As the categorization results in a single category estimate, errors become more severe. Thus, the computation should be more accurate than during retrieval, where many true positives may well be mixed with a few false positives. An accurate calculation requires normalized feature vectors to have the different features comparable in a comparable range. Further, the thresholds must not be used too restrictively (i.e. cutting off results with lower similarity) to acknowledge relevant images with a slightly lower similarity. To achieve this, second normalization step according to the thresholds (see eq. (3.31)) proved to be vital.

In direct comparison of the final case study (section 6.6) to the supervised learning (section 6.4), the recognition performance dropped clearly. This seems mostly to be caused by sample objects with many differences to the training set used. It is found, that the combination of feature vectors used in the prototype can in average compete with each training on a single feature vector from the reference system. In the reference system, each feature vector performed differently well for each category.

In comparison, the relatively weak feature vectors of the proposed system are in combination as good as the average of the reference system. The recognition rate has been achieved with a limited set of positive and negative examples (i.e. a maximum of 20 distinct clusters for a category) and without further low-level optimization of the features themselves.

Chapter 7.

Conclusion

In order to investigate the research hypothesis from section 1.1 several case studies have been carried out (chapter 6). Each single case study is required to support parts of the methodology in chapter 3. The final evaluation has been carried out by comparing the expressiveness of the descriptors learnt in a leave-one-object-out scenario to a traditional machine learning approach (section 6.6).

It is found, that the query language used can be as powerful as a feature vector derived from a traditional machine learning algorithm. A Boolean query descriptor can be used to describe a single category relatively precisely. Further, it does not matter, if the category to be learnt is located in multiple disjunct clusters in the feature space, as the Boolean join can handle it easily. The weakest part of the proposed learning approach are the feature vectors themselves. If they are not powerful enough to capture distinctive properties for a category, the whole learning process for the given category cannot be successful. Yet, as long as there is a feature vector available giving a slight improvement towards a random search, it is spotted by the algorithm and is used to improve the results.

In conclusion: a high-level mapping of semantics (i.e. a semantic category) can only be as good as the low-level feature vectors are capable to distinguish positive samples from unrelated categories.

7.1. Achieved

The proposed query language seems to be powerful enough to describe most common retrieval tasks. The search for virtually any feature vector is possible as long as they can be mapped to a normalized similarity of $[0, 1]$ between two feature vectors of the same kind. Further, merging multiple sub queries is achieved by boolean operators

and parameters. Complex queries containing low-level features only, are a possible way of expressing higher-level semantics, at least to a certain extent. The query descriptors generated by the learning algorithm are mapping semantic categories to the most efficient image features available.

The tests indicated that the feature vector modules applied in this thesis are too few and not diverse enough to fully compete with recent machine learning systems. Being independent from the feature vectors used, the results are still promising. The results indicate that by adding further plug-ins the recognition rate of the system will be improved without changing the underlying methodology. The learning algorithm has been developed as a tool to automatically generate meaningful queries. This way, it was possible to evaluate the query language in depth. Further, several techniques applied during the development and evaluation of the learning algorithm are useful in retrieval, too. One example is the feature vector normalization. Is important to make features of different kinds comparable. This allows for a more efficient combination of arbitrary features into a single result.

Traditional machine learning approaches often encode the derived knowledge in complex and difficult to understand data. For SVMs this could be concatenating several feature vectors into a single big data set. ANNs usually create a set of weights with no obvious meaning to human perception. Unlike that, the presented DT approach generates a set of comprehensible clauses. This allows experienced users to manually modify the classification queries, e.g. to remove a set of undesired positives.

7.2. Future Work

The theoretical foundations of the thesis indicate several opportunities for further related research, e.g.:

- Development of an intuitive CBIR user interface
- Support for spatial information integrated into the query language handle local objects
- Integration of an efficient image segmentation algorithm
- Implementation of powerful feature vectors, such as shape matching
- Focus on higher scalability, e.g. stronger use of index structures and heuristics

- Application of other efficient learning techniques to build query descriptors
- Use of query descriptors to replace keywords in CBIR queries or to label images with unknown content
- Improvement of the Boolean ranking mechanisms, e.g. by applying Bayesian combination rules

Bibliography

[1] A9.com. OpenSearch, 2006. URL `http://www.opensearch.org/Home`.

[2] S. Agarwal and D. Roth. Learning a sparse representation for object detection. In *Proceedings of the European Conference on Computer Vision*, volume 4, pages 113–130, Copenhagen, Denmark, May 2002. Springer-Verlag.

[3] S. Agarwal, A. Awan, and D. Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490, November 2004.

[4] Faruq A. Al-Omari and Mohammad A. Al-Jarrah. Query by image and video content: a colored-based stochastic model approach. *Data Knowl. Eng.*, 52(3):313–332, 2005. ISSN 0169-023X. doi: http://dx.doi.org/10.1016/j.datak.2004.06.008.

[5] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 583–594, New York, NY, USA, 2004. ACM. ISBN 1-58113-844-X. doi: http://doi.acm.org/10.1145/988672.988751.

[6] Apache Software Foundation. Apache Lucene, 2006. URL `http://lucene.apache.org/`.

[7] Apache Software Foundation. Apache Lucene Query Syntax, 2008. URL `http://lucene.apache.org/java/2_3_1/queryparsersyntax.html`.

[8] J. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Gorowitz, R. Humphrey, R. Jain, and C. Shu. Virage image search engine: an open framework for image management. In I. K. Sethi and R. C. Jain, editors, *SPIE, Storage and Retrieval for Image and Video Databases IV*, pages 76–87, February 1996.

[9] M. Beigi, A. B. Benitez, and S. F. Chang. MetaSEEk: a content-based metasearch engine for images. In I. K. Sethi and R. C. Jain, editors, *Storage and Retrieval*

for Image and Video Databases VI, volume 3312 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 118–128, dec 1997. doi: 10.1117/12.298436.

[10] S. Belongie, J. Malik, and J. Puzicha. Matching Shapes. In *International Conference on Computer Vision (ICCV)*, 2001.

[11] S. Belongie, J. Malik, and J. Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.993558.

[12] A. B. Benitez, M. Beigi, and S.-F. Chang. Using relevance feedback in content-based image metasearch. *IEEE Internet Computing*, 2(4):59–69, July 1998.

[13] J. Berens, G.D. Finlayson, and G. Qiu. Image indexing using compressed colour histograms. *Vision, Image and Signal Processing, IEE Proceedings* -, 147(4):349–355, Aug 2000. ISSN 1350-245X. doi: 10.1049/ip-vis:20000630.

[14] Andreas D. Blaser and Max. J. Egenhofer. A visual tool for querying geographic databases. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, pages 211–216, New York, NY, USA, 2000. ACM. ISBN 1-58113-252-2. doi: http://doi.acm.org/10.1145/345513.345318.

[15] Martijn Bosma, Remco C. Veltkamp, and Frans Wiering. Muugle: A Modular Music Information Retrieval Framework. In *ISMIR*, pages 330–331, 2006.

[16] Jean Marie Buijs and Michael S. Lew. Visual Learning of Simple Semantics in ImageScape. In *VISUAL '99: Proceedings of the Third International Conference on Visual Information and Information Systems*, pages 131–138, London, UK, 1999. Springer-Verlag. ISBN 3-540-66079-8.

[17] D. Calcinelli and M. Mainguenaud. Cigales, a Visual Query Language for a Geographical Information System - the User Interface. *Journal of Visual Languages and Computing*, 5:113–132, 1994.

[18] K. Selçuk Candan, Eric Lemar, Eric Lemar, V. S. Subrahmanian, and V. S. Subrahmanian. View management in multimedia databases. *The VLDB Journal*, 9 (2):131–153, 2000. ISSN 1066-8888. doi: http://dx.doi.org/10.1007/PL00010673.

[19] A. F. Cardenas, I. T. Ieong, R. Barker, R. K. Taira, and C. M. Breant. The Knowledge-Based Object-Oriented PICQUERY+ Language. *IEEE Trans. on Knowl. and Data Eng.*, 5(4):644–657, 1993. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/69.234776.

[20] Don Chamberlin, James Clark, Daniela Florescu, Jonathan Robie, Jérôme Siméon, and Mugur Stefanescu. XQuery 1.0: An XML Query Language, 2001. W3C Working Draft 07 June 2001.

[21] Donald D. Chamberlin and Raymond F. Boyce. SEQUEL: A structured English query language. In *FIDET '74: Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pages 249–264, New York, NY, USA, 1974. ACM. doi: http://doi.acm.org/10.1145/800296.811515.

[22] Krishna Chandramouli and Ebroul Izquierdo. Image Classification Using Self Organizing Feature Maps and Particle Swarm Optimization. In *7th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS '06)*, pages 313–316, 2006.

[23] Shih-Fu Chang, John R. Smith, Mandis Beigi, and Ana Benitez. Visual information retrieval from large distributed online repositories. *Commun. ACM*, 40(12):63–71, 1997. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/265563.265573.

[24] Cheng-Chieh Chiang, Li-Wei Chan, Yi-Ping Hung, and Greg C. Lee. Content-based object movie retrieval and relevance feedbacks. *EURASIP J. Adv. Signal Process*, 2007(2):24–24, 2007. ISSN 1110-8657. doi: http://dx.doi.org/10.1155/2007/89691.

[25] Paul Clough, Michael Grubinger, Thomas Deselaers, Allan Hanbury, and Henning Müller. *Evaluation of Multilingual and Multi-modal Information Retrieval*, volume 4730/2007, chapter Overview of the ImageCLEF 2006 Photographic Retrieval and Object Annotation Tasks, pages 579–594. Springer, 2007. doi: 10.1007/11878773.

[26] Tammara T. A. Combs and Benjamin B. Bederson. Does zooming improve image browsing? In *DL '99: Proceedings of the fourth ACM conference on Digital libraries*, pages 130–137, New York, NY, USA, 1999. ACM. ISBN 1-58113-145-3. doi: http://doi.acm.org/10.1145/313238.313286.

[27] CompuServe Incorporated. Graphics Interchange Format. Specification, 1990. URL `http://www.w3.org/Graphics/GIF/spec-gif89a.txt`.

[28] Ritendra Datta, Jia Li, and James Z. Wang. Content-based image retrieval: approaches and trends of the new age. In *MIR '05: Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 253–262, New York, NY, USA, 2005. ACM. ISBN 1-59593-244-5. doi: http://doi.acm.org/10.1145/1101826.1101866.

[29] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. *Computer Vision ECCV 2006*, volume 3953/2006 of *Lecture Notes in Computer Science*, chapter Studying Aesthetics in Photographic Images Using a Computational Approach, pages 288–301. Springer Berlin / Heidelberg, July 2006. doi: 10.1007/11744078.

[30] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Transactions on Computing Surveys*, 40(2):1–60, April 2008.

[31] David Duce, editor. Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E). Technical report, World Wide Web Consortium (W3C), November 2003. URL `http://www.w3.org/TR/2003/REC-PNG-20031110/`. W3C Recommendation.

[32] M.N. Do and M. Vetterli. Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance. *Image Processing, IEEE Transactions on*, 11(2):146–158, Feb 2002. ISSN 1057-7149. doi: 10.1109/83.982822.

[33] N.D. Doulamis, A.D. Doulamis, and S.D. Kollias. A neural network approach to interactive content-based retrieval of video databases. *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, 2:116–120, 1999. doi: 10.1109/ICIP.1999.822866.

[34] J.P. Eakins and M.E. Graham. Content-based Image Retrieval. A Report to the JISC Technology Applications Programme. Technical report, University of Northumbria at Newcastle, January 1999. URL `http://www.jisc.ac.uk/uploaded_documents/jtap-039.doc`.

[35] Earth Resource Mapping Pty Ltd. Using and distributing ECW V2.0 wavelet compressed imagery. White Paper, 1999.

[36] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results.

http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html, 2007.

[37] Ronald Fagin. Combining fuzzy information from multiple systems (extended abstract). In *PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 216–226, New York, NY, USA, 1996. ACM. ISBN 0-89791-781-2. doi: http://doi.acm.org/10.1145/237661.237715.

[38] Vincenzo Del Fatto, Luca Paolino, and Fabio Pittarello. A usability-driven approach to the development of a 3D web-GIS environment. *Journal of Visual Languages and Computing*, 18:280314, 2007. doi: 10.1016/j.jvlc.2007.02.007.

[39] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27: 861874, 2006.

[40] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In *International Conference on Computer Vision and Pattern Recognition (CVPR'04)*, 2004. Workshop on Generative-Model Based Vision.

[41] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, Madison, Wisconsin, June 2003.

[42] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[43] David Frohlich, Allan Kuchinsky, Celine Pering, Abbe Don, and Steven Ariss. Requirements for photoware. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 166–175, New York, NY, USA, 2002. ACM. ISBN 1-58113-560-2. doi: http://doi.acm.org/10.1145/587078.587102.

[44] Sharon R. Garber and Mitch B. Grunes. The art of search: a study of art directors. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 157–163, New York, NY, USA, 1992. ACM. ISBN 0-89791-513-5. doi: http://doi.acm.org/10.1145/142750.142780.

[45] Elisa Drelie Gelasca, Pratim Ghosh, Emily Moxley, Joriz De Guzman, JieJun Xu, Zhiqiang Bi, Steffen Gauglitz, Amir M. Rahimi, and B. S. Manjunath. CORTINA: Searching a 10 Million + Images Database. In *Very Large Data Base Endowment*, 2007.

[46] Google, Inc. Google Query Syntax, 2008. URL `http://code.google.com/apis/base/query-lang-spec.html`.

[47] Google, Inc. Picasa, 2008. URL `http://picasa.google.com/`.

[48] G. Griffin, A. Holub, and P. Perona. Caltech-256 Object Category Dataset. Technical Report 7694, California Institute of Technology, 2007. URL `http://authors.library.caltech.edu/7694`.

[49] N. Gunther and G. Beretta. Benchmark for image retrieval using distributed systems over the Internet: BIRDS-I, 2001. URL `citeseer.ist.psu.edu/gunther00benchmark.html`.

[50] Amarnath Gupta and Ramesh Jain. Visual information retrieval. *Commun. ACM*, 40(5):70–79, 1997. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/253769.253798.

[51] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *ACM SIGMOD Conf. on the Management of Data*, pages 47–57, 1984.

[52] Alexander G. Hauptmann and Michael G. Christel. Successful approaches in the TREC video retrieval evaluations. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 668–675, New York, NY, USA, 2004. ACM. ISBN 1-58113-893-8. doi: http://doi.acm.org/10.1145/1027527.1027681.

[53] S. Hibino and E.A. Rundensteiner. A Visual Query Language for Identifying Temporal Trends in Video Data. *iw-mmdbms*, 00:0074, 1995. doi: http://doi.ieeecomputersociety.org/10.1109/MMDBMS.1995.520425.

[54] Chu-Hong Hoi and M.R. Lyu. Group-based relevance feedback with support vector machine ensembles. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 3:874–877, Aug 2004. ISSN 1051-4651. doi: 10.1109/ICPR.2004.1334667.

[55] David F. Huynh, Steven M. Drucker, Patrick Baudisch, and Curtis Wong. Time quilt: scaling up zoomable photo browsers for large, unstructured photo collections. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1937–1940, New York, NY, USA, 2005. ACM. ISBN 1-59593-002-7. doi: http://doi.acm.org/10.1145/1056808.1057061.

[56] IPTC [International Press Telecommunications Council]. IPTC Standard - Photo Metadata 2008. Specification, 2008.

[57] Yoshiharu Ishikawa, Ravishankar Subramanya, and Christos Faloutsos. MindReader: Querying Databases Through Multiple Examples. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 218–227. Morgan Kaufmann, 1998. ISBN 1-55860-566-5.

[58] Charles E. Jacobs, Adam Finkelstein, and David H. Salesin. Fast Multiresolution Image Querying. *Computer Graphics*, 29(Annual Conference Series):277–286, 1995. URL `citeseer.ist.psu.edu/jacobs95fast.html`.

[59] Anil Jain, Karthik Nandakumar, and Arun Ross. Score normalization in multimodal biometric systems. *Pattern Recognition*, 38(12):2270–2285, 2005. ISSN 0031-3203. doi: DOI:10.1016/j.patcog.2005.01.012. URL `http://www.sciencedirect.com/science/article/B6V14-4G0DDW4-1/2/d922960ee7ed8928744113dd9494d37a`.

[60] Kalervo Järvelin, Timo Niemi, and Airi Salminen. The visual query language CQL for transitive and relational computation. *Data & Knowledge Engineering*, 35:39–51, 2000.

[61] Yohan Jin, Latifur Khan, Lei Wang, and Mamoun Awad. Image annotations by combining multiple evidence & wordNet. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 706–715, New York, NY, USA, 2005. ACM. ISBN 1-59593-044-2. doi: http://doi.acm.org/10.1145/1101149.1101305.

[62] Jean-Michel Jolion. *Principles of Visual Information Retrieval*, chapter Feature similarity, pages 121–143. Advances in Pattern Recognition. Springer-Verlag, London, UK, 2001. ISBN 1-85233-381-2.

[63] Dhiraj Joshi, Ritendra Datta, Ziming Zhuang, W. P. Weiss, Marc Friedenberg, Jia Li, and James Z. Wang. PARAgrab: a comprehensive architecture for web image management and multimodal querying. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 1163–1166. VLDB Endowment, 2006.

[64] Daniel A. Keim and Vincent Lum. Visual query specification in a multimedia database system. In *VIS '92: Proceedings of the 3rd conference on Visualization '92*, pages 194–201, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-8186-2896-0.

[65] Deok-Hwan Kim and Chin-Wan Chung. QCluster: relevance feedback using adaptive clustering for content-based image retrieval. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 599–610, New York, NY, USA, 2003. ACM. ISBN 1-58113-634-X. doi: http://doi.acm.org/10.1145/872757.872829.

[66] Longin Jan Latecki and Rolf Lakämper. Shape Similarity Measure Based on Correspondence of Visual Parts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(10): 1185–1190, 2000. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/34.879802.

[67] B. Leibe, A. Leonardis, and B. Schiele. Combined object categorization and segmentation with an implicit shape model. In *Proceedings of the Workshop on Statistical Learning in Computer Vision*, Prague, Czech Republic, May 2004.

[68] Bastian Leibe and Bernt Schiele. Analyzing Appearance and Contour Based Methods for Object Categorization. In *International Conference on Computer Vision and Pattern Recognition (CVPR'03)*, June 2003.

[69] A. Leuski and J. Allan. Improving interactive retrieval by combining ranked lists and clustering. In *Proceedings of RIAO'2000*, pages 665–681, 2000. URL `citeseer.ist.psu.edu/leuski00improving.html`.

[70] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, 2006. ISSN 1551-6857. doi: http://doi.acm.org/10.1145/1126004.1126005.

[71] Hao Liu, Xing Xie, Xiaoou Tang, Zhi-Wei Li, and Wei-Ying Ma. Effective browsing of web image search results. In *MIR '04: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 84–90, New York, NY, USA, 2004. ACM. ISBN 1-58113-940-3. doi: http://doi.acm.org/10.1145/1026711.1026726.

[72] LizardTech, Inc. MrSID Technology Primer, 2004. URL `http://www.lizardtech.com/files/geo/techinfo/MrSID_Tech_Primer.pdf`.

[73] Fariborz Mahmoudi, Jamshid Shanbehzadeh, Amir-Masoud Eftekhari-Moghadam, and Hamid Soltanian-Zadeh. Image retrieval based on shape similarity by edge orientation autocorrelogram. *Pattern Recognition*, 36:1725–1736, 2003.

[74] B.S. Manjunath, J.-R. Ohm, V.V. Vasudevan, and A. Yamada. Color and texture descriptors. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):703–715, Jun 2001. ISSN 1051-8215. doi: 10.1109/76.927424.

[75] J.M. Martinez, R. Koenen, and F. Pereira. MPEG-7: The Generic Multimedia Content Description Standard, Part 1. *IEEE MultiMedia*, 09(2):78–87, 2002. ISSN 1070-986X. doi: http://doi.ieeecomputersociety.org/10.1109/MMUL.2002.10016.

[76] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: An On-line Lexical Database. *International Journal of Lexicography*, 3(4):235–244, 1990. doi: 10.1093/ijl/3.4.235.

[77] H. Müller, W. Müller, S. Marchand-Maillet, T. Pun, and D.M. Squire. Strategies for positive and negative relevance feedback in image retrieval. *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, 1:1043–1046 vol.1, 2000. doi: 10.1109/ICPR.2000.905650. URL `citeseer.ist.psu.edu/327327.html`.

[78] Henning Müller, Wolfgang Müller, David McG. Squire, Zoran Pečenović, Stéphane Marchand-Maillet, and Thierry Pun. An Open Framework for Distributed Multimedia Retrieval. In *Recherche d'Informations Assistée par Ordinateur (RIAO'2000) Computer-Assisted Information Retrieval, Paris, France*, April 2000.

[79] Henning Müller, Wolfgang Müller, Stéphane Marchand-Maillet, Thierry Pun, and David McG. Squire. A Framework for Benchmarking in CBIR. *Multimedia Tools and Applications*, 21(1):55–73, September 2003. doi: 10.1023/A:1025034215859.

[80] Henning Müller, Paul Clough, William Hersh, Thomas Deselaers, Thomas Lehmann, and Antoine Geissbuhler. Evaluation axes for medical image retrieval systems: the imageCLEF experience. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 1014–1022, New York, NY, USA, 2005. ACM. ISBN 1-59593-044-2. doi: http://doi.acm.org/10.1145/1101149.1101358.

[81] Wolfgang Müller, Henning Müller, Stéphane Marchand-Maillet, Thierry Pun, David McG. Squire, Zoran Pečenović, Christoph Giess, and Arjen P. de Vries. MRML: A Communication Protocol for Content-Based Image Retrieval. In *International Conference on Visual Information Systems (Visual 2000), Lyon, France*, November 2000.

[82] H. Murase and S.K. Nayar. Visual Learning and Recognition of 3D Objects from Appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[83] S. Nepal, M.V. Ramakrishna, and J.A. Thom. A fuzzy object query language (FOQL) for image databases. *Database Systems for Advanced Applications, 1999. Proceedings., 6th International Conference on*, pages 117–124, 1999. doi: 10.1109/DASFAA.1999.765743.

[84] Surya Nepal and M.V. Ramakrishna. Query Processing Issues in Image(Multimedia) Databases. *International Conference on Data Engineering*, 00:22–29, 1999. ISSN 1063-6382. doi: http://doi.ieeecomputersociety.org/10.1109/ICDE.1999.754894.

[85] W. Niblack, X. Zhu, J. Hafner, T. Breuel, D. Ponceleón, D. Petkovic, M. Flickner, E. Upfal, S. Nin, S. Sull, B. Dom, B.-L. Yeo, S. Srinivasan, D. Zivkovic, and M. Penner. Updates to the QBIC system. *Retrieval for Image and Video Databases VI*, 3312:150–161, 1998.

[86] A. Opelt and A. Pinz. Object localization with boosting and weak supervision for generic object recognition. In *Proceedings of the 14th Scandinavian Conference on Image Analysis (SCIA)*, 2005.

[87] A. Opelt, A. Pinz, M. Fussenegger, and P. Auer. Generic object recognition with boosting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28 (3):416–431, March 2006. ISSN 0162-8828. doi: 10.1109/TPAMI.2006.54.

[88] Luca Paolino, Genoveffa Tortora, Monica Sebillo, Giuliana Vitiello, and Robert Laurini. Phenomena: a visual query language for continuous fields. In *GIS '03: Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 147–153, New York, NY, USA, 2003. ACM. ISBN 1-58113-730-3. doi: http://doi.acm.org/10.1145/956676.956696.

[89] Raoul Pascal Pein. Multi-Modal Image Retrieval - A Feasibility Study, 2005. Diplomarbeit.

[90] Raoul Pascal Pein. Hot-Pluggable Multi-Feature Search Engine. Master's thesis, Hamburg University of Applied Sciences, 2008.

[91] Raoul Pascal Pein and Joan Lu. Multi-feature query language for image classification. In *International Conference on Computational Science, ICCS 2010*. Elsevier, 2010. Accepted for publication.

[92] Raoul Pascal Pein and Zhongyu Lu. Content Based Image Retrieval by Combining Features and Query-By-Sketch. In Hamid R. Arabnia and Ray R. Hashemi, editors, *International Conference on Information & Knowledge Engineering (IKE)*, pages 49–55, Las Vegas, USA, June 2006. CSREA Press. ISBN 1-60132-003-5.

[93] Raoul Pascal Pein and Zhongyu Lu. A Flexible Image Retrieval Framework. In Yong Shi, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *International Conference on Computational Science (3)*, volume 4489 of *Lecture Notes in Computer Science*, pages 754–761. Springer, may 2007. ISBN 978-3-540-72587-9. doi: 10.1007/978-3-540-72588-6_124.

[94] Raoul Pascal Pein, Milton Amador, Joan Lu, and Wolfgang Renz. Using CBIR and Semantics in 3D-Model Retrieval. In *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*, pages 173–178, July 2008. doi: 10.1109/CIT.2008.4594669.

[95] Raoul Pascal Pein, Joan Lu, and Wolfgang Renz. An Extensible Query Language for Content Based Image Retrieval based on Lucene. In *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*, pages 179–184, July 2008. doi: 10.1109/CIT.2008.4594670.

[96] Raoul Pascal Pein, Joan Lu, and Wolfgang Renz. An Extensible Query Language for Content Based Image Retrieval. *The Open Information Systems Journal*, 17: 81–97, July 2008. doi: 10.1109/CIT.2008.4594670. Bentham Open.

[97] W.S. Peng and N. DeClaris. Heuristic similarity measure characterization for content-based image retrieval. *Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on*, 1: 7–12, Oct 1997. doi: 10.1109/ICSMC.1997.625710.

[98] Kriengkrai Porkaew, Kaushik Chakrabarti, and Sharad Mehrotra. Query Refinement for Multimedia Similarity Retrieval in MARS. In *Proceedings of ACM Multimedia*, pages 235–238, 1999.

[99] P. Praks, E. Izquierdo, and R. Kucera. The sparse image representation for automated image retrieval. In *IEEE International Conference on Image Processing*, pages 25–28, October 2008.

[100] Till Quack, Ullrich Mönich, Lars Thiele, and B. S. Manjunath. Cortina: a system for large-scale, content-based web image retrieval. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 508–511, New York, NY, USA, 2004. ACM. ISBN 1-58113-893-8. doi: http://doi.acm.org/10.1145/1027527.1027650.

[101] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. doi: 10.1007/BF00116251. Computer Science.

[102] J. Quinonero-Candela, I. Dagan, B. Magnini, and F. d'Alch Buc, editors. *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment*, volume 3944/2006 of *Lecture Notes in Computer Science*, chapter The 2005 PASCAL Visual Object Classes Challenge, pages 117–176. Springer-Verlag, 2006. doi: 10.1007/11736790. URL `http://eprints.pascal-network.org/archive/00001212/01/voc11.uk.pdf`.

[103] M. V. Ramakrishna, S. Nepal, and P. K. Srivastava. A heuristic for combining fuzzy results in multimedia databases. In *ADC '02: Proceedings of the 13th Australasian database conference*, pages 141–144, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc. ISBN 0-909925-83-6.

[104] David Riecks. "IPTC Core" Schema for XMP - Version 1.0. Technical report, International Press Telecommunications Council, 2005.

[105] Joseph John Rocchio. Relevance feedback in information retrieval. In Gerard Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice-Hall, Englewood Cliffs, NJ, USA, 1971.

[106] Kerry Rodden. *Evaluating similarity-based visualisations as interfaces for image browsing*. PhD thesis, University of Cambridge, September 2002.

[107] Kerry Rodden and Kenneth R. Wood. How do people manage their digital photographs? In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 409–416, New York, NY, USA, 2003. ACM. ISBN 1-58113-630-7. doi: http://doi.acm.org/10.1145/642611.642682.

[108] Kerry Rodden, Wojciech Basalaj, David Sinclair, and Kenneth Wood. Does organisation by similarity assist image browsing? In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 190–197, New York, NY, USA, 2001. ACM. ISBN 1-58113-327-8. doi: http://doi.acm.org/10.1145/365024.365097.

[109] Yong Rui and Thomas Huang. Optimizing Learning in Image Retrieval. *cvpr*, 01:1236, 2000. ISSN 1063-6919. doi: http://doi.ieeecomputersociety.org/10.1109/CVPR.2000.855825.

[110] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. LabelMe: A Database and Web-Based Tool for Image Annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, 2008. ISSN 0920-5691. doi: http://dx.doi.org/10.1007/s11263-007-0090-8.

[111] Simone Santini and Ramesh Jain. Similarity Measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):871–883, 1999. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.790428.

[112] Bertrand Le Saux and Nozha Boujemaa. Unsupervised Robust Clustering for Image Database Categorization. *icpr*, 01:10259, 2002. ISSN 1051-4651. doi: http://doi.ieeecomputersociety.org/10.1109/ICPR.2002.1044678.

[113] Robert E. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5: 197–227, 1990.

[114] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):Jonathan BaxterNicol'o Cesa-Bianchi, December 1999. doi: 10.1023/A:1007614523901.

[115] B. Schiele and J.L. Crowley. Recognition without correspondence using multidimensional receptive field histograms. *International Journal of Computer Vision*, 36(1):31–52, 2000.

[116] Jan Schietse, John P. Eakins, and Remco C. Veltkamp. Practice and challenges in trademark image retrieval. In *CIVR '07: Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 518–524, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-733-9. doi: http://doi.acm.org/10.1145/1282280.1282355.

[117] P. Schnitzspan, M. Fritz, S. Roth, and B. Schiele. Discriminative Structure Learning of Hierarchical Representations for Object Detection. In *Computer Vision and Pattern Recognition (CVPR)*, Miami, USA, 2009.

[118] E. Di Sciascio, G. Mingolla, and M. Mongiello. Content-based image retrieval over the web using query by sketch and relevance feedback. In D. P. Huijsmans and A. W. M. Smeulders, editors, *Visual Information and Information Systems*, pages 123–130, June 1999.

[119] Thomas Seidl and Hans-Peter Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 506–515, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-470-7.

[120] G. Sheikholeslami, S. Chatterjee, and A. Zhang. NeuroMerge: an approach for merging heterogeneous features in content-based image retrieval systems. *Multi-Media Database Management Systems, 1998. Proceedings. International Workshop on*, pages 106–113, Aug 1998. doi: 10.1109/MMDBMS.1998.709516.

[121] B. Shneiderman and H. Kang. Direct annotation: a drag-and-drop strategy for labeling photos. *Information Visualization, 2000. Proceedings. IEEE International Conference on*, pages 88–95, 2000. doi: 10.1109/IV.2000.859742.

[122] Ben Shneiderman, Benjamin B. Bederson, and Steven M. Drucker. Find that photo!: interface strategies to annotate, browse, and share. *Commun. ACM*, 49(4): 69–71, 2006. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1121949.1121985.

[123] Alan F. Smeaton, Paul Over, and Wessel Kraaij. Evaluation campaigns and TRECVid. In *MIR '06: Proceedings of the 8th ACM International Workshop*

on Multimedia Information Retrieval, pages 321–330, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-495-2. doi: http://doi.acm.org/10.1145/1178677.1178722.

[124] Arnold W.M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-Based Image Retrieval at the End of the Early Years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.895972.

[125] J. R. Smith and S.-F. Chang. Querying by color regions using the VisualSEEk content-based visual query system. In M. T. Maybury, editor, *Intelligent Multimedia Information Retrieval*. AAAI Press, 1997.

[126] John R. Smith. MARVEL: Multimedia Analysis and Retrieval System. Whitepaper, Intelligent Information Management Dept., IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 USA, 2004.

[127] Christoph Stamm. PGF - A new progressive file format for lossy and lossless image compression. Technical report, ETH Zurich, Institute of Theoretical Computer Science, 2002. URL `http://www.libpgf.org/uploads/media/PGF_stamm_wscg02.pdf`.

[128] Bongwon Suh, Haibin Ling, Benjamin B. Bederson, and David W. Jacobs. Automatic thumbnail cropping and its effectiveness. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 95–104, New York, NY, USA, 2003. ACM. ISBN 1-58113-636-6. doi: http://doi.acm.org/10.1145/964696.964707.

[129] M. J. Swain and D.H. Ballard. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[130] David S. Taubman and Michael W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Norwell, MA, USA, 2001. ISBN 079237519X.

[131] Leonid Taycher, Marco La Cascia, and Stan Sclaroff. Image Digestion and Relevance Feedback in the ImageRover WWW Search Engine. In *2nd International Conference on Visual Information Systems*, pages 85–94, December 1997.

[132] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 762–769, Washington, DC, June 2004.

[133] Ricardo S. Torres, Celmar G. Silva, Claudia B. Medeiros, and Heloisa V. Rocha. Visual structures for image browsing. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 49–55, New York, NY, USA, 2003. ACM. ISBN 1-58113-723-0. doi: http://doi.acm.org/10.1145/956863.956874.

[134] C. Town and D. Sinclair. Ontological query language for content based image retrieval. In *Content-Based Access of Image and Video Libraries, 2001. (CBAIVL 2001). IEEE Workshop on*, pages 75–80, 14 Dec. 2001. doi: 10.1109/IVL.2001.990859.

[135] M. Turk and A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3:71–86, 1991.

[136] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[137] Gregory K. Wallace. The JPEG Still Picture Compression Standard. Technical report, Multimedia Engineering Digital Equipment Corporation Maynard, Massachusetts, 1991.

[138] James Z. Wang, Jia Li, and Gio Wiederhold. SIMPLIcity: Semantics-Sensitive Integrated Matching for Picture LIbraries. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(9):947–963, 2001. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/34.955109.

[139] James Z. Wang, Nozha Boujemaa, Alberto Del Bimbo, Donald Geman, Alexander G. Hauptmann, and Jelena Tesić. Diversity in multimedia information retrieval research. In *MIR '06: Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pages 5–12, New York, NY, USA, 2006. ACM. ISBN 1-59593-495-2. doi: http://doi.acm.org/10.1145/1178677.1178681.

[140] James Z. Wang, Nozha Boujemaa, and Yixin Chen. High diversity transforms multimedia information retrieval into a cross-cutting field: report on the 8th Workshop

on Multimedia Information Retrieval. *SIGMOD Rec.*, 36(1):57–59, 2007. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/1276301.1276315.

[141] Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, and Vladimir Vapnik. Feature Selection for SVMs. In *NIPS*, pages 668–674, 2000. URL `citeseer.ist.psu.edu/article/weston01feature.html`.

[142] Kevin Woods, Kevin Bowyer, and W. Philip Kegelmeyer Jr. Combination of multiple classifiers using local accuracy estimates. In *CVPR '96: Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96)*, page 391, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7258-7.

[143] Bo Wu and R. Nevatia. Cluster boosted tree classifier for multi-view, multi-pose object detection. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct. 2007. doi: 10.1109/ICCV.2007.4409006.

[144] Yi Wu, Edward Y. Chang, Kevin Chen-Chuan Chang, and John R. Smith. Optimal multimodal fusion for multimedia data analysis. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 572–579, New York, NY, USA, 2004. ACM. ISBN 1-58113-893-8. doi: http://doi.acm.org/10.1145/1027527.1027665.

[145] Yahoo! Inc. Flickr - Photo Sharing, 2006. URL `http://www.flickr.com/`.

[146] Yufei Yuan and Michael J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69(2):125–139, 1995. ISSN 0165-0114. doi: DOI:10.1016/0165-0114(94)00229-Z. URL `http://www.sciencedirect.com/science/article/B6V05-4007D5X-C/2/62fa7de5b67d27ad22a38a1f4e6ba0e6`.

[147] Lotfi A. Zadeh. *Fuzzy sets*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996. ISBN 981-02-2422-2.

[148] Q. Zhang and E. Izquierdo. A New Approach to Image Retrieval in a Multi-Feature Space. In *International Workshop on Image Analysis for Multimedia Interactive Services*, April 2006.

[149] Rong Zhao and William I. Grosky. Bridging the semanitic gap in image retrieval. *Distributed multimedia databases: techniques & applications*, pages 14–36, 2002.

[150] Xiang Sean Zhou and Thomas S. Huang. Comparing discriminating transformations and SVM for learning during multimedia retrieval. In *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pages 137–146, New York, NY, USA, 2001. ACM. ISBN 1-58113-394-4. doi: http://doi.acm.org/10.1145/500141.500163.

[151] Xiang Sean Zhou and Thomas S. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, 8(6):536–544, April 2003. doi: 10.1007/s00530-002-0070-3.

Appendix A.

Definitions

tp True Positives

fp False Postives

tn True Negatives

fn False Negatives

I set of all images i in a repository

R set of all relevant images (Positives)

R_x set of all relevant images (Positives) of category x

N set of all non-relevant images (Negatives)

E evaluation set

T training set

F feature vector set

f feature vector

i single image

ι set of retrieved images

$r(q)$ result set of query q

q query

π precision

ρ recall

ν node in decision tree

$c^{[+|-]}$ boolean clause $[MUST|MUST\_NOT]$

$C^{[+|-]}$ set of boolean clauses $[MUST|MUST\_NOT]$

t term

$f_x(i)$ feature vector x of image i

$s(f(i_1), f(i_2))$ similarity of i_1 and i_2 based on feature vector f

Appendix B.

Descriptors

This table lists all query descriptors generated in the supervised learning on the ETH-80 images (section 6.4). The threshold and boost values are cut off after 3 decimal places.

apple	RGB:124^0.978@0.918 H:237^0.819@0.975 RGB:215^0.680@0.915 RGB:281^0.655@0.971 RGB:264^0.393@0.981 RGB:68^0.311@0.989 RGB:76^0.311@0.993
car	W:684^0.898@0.951 SH:556^0.804@0.967 H:460^0.720@0.944 SH:634^0.658@0.961 SH:525^0.583@0.962 W:700^0.583@0.954 SH:751^0.542@0.978 W:654^0.496@0.954 W:501^0.459@0.933 SH:540^0.439@0.973 W:422^0.368@0.951 W:427^0.311@0.956 W:442^0.311@0.954 SH:484^0.278@0.972 W:705^0.278@0.950 W:438^0.242@0.957 SH:620^0.242@0.960 SH:435^0.222@0.986 H:667^0.222@0.978 H:703^0.213@0.987
cow	(W:856@0.953 -SH:1300@0.973 -W:2130@0.969 -H:2245@0.998)^0.665 SH:1085^0.667@0.971 W:979@0.951^0.592 SH:845^0.526@0.970 H:1127^0.483@0.988 W:1006^0.487@0.950 (W:1018@0.951 -RGB:813@0.969)^0.416 W:1058^0.417@0.901 SH:1153^0.435@0.953 W:846^0.417@0.951 SH:873^0.379@0.972 SH:849^0.389@0.984 RGB:1209^0.404@0.997 W:909^0.354@0.958 W:1168^0.311@0.956 W:875^0.341@0.955 W:852^0.278@0.957 SH:1054^0.278@0.979 RGB:947^0.242@0.998 W:1000^0.242@0.951
cup	H:1244^0.982@0.844 H:1572^0.720@0.985 W:1635^0.672@0.953 W:1299^0.529@0.959 RGB:1463^0.459@0.978 RGB:1603^0.459@0.985 SH:1454^0.340@0.858 RGB:1293^0.278@0.974
dog	(W:1963@0.888 -SH:2193@0.975 -SH:2348@0.981)^0.721 (SH:2038@0.962 -SH:1104@0.997 -H:2121@0.995)^0.683 (H:2004@0.990 -RGB:2100@0.998)^0.582 SH:2033^0.577@0.966 H:1804^0.544@0.992 W:1913^0.544@0.923 W:1707^0.528@0.943 SH:1650^0.463@0.959 SH:1809^0.463@0.972 W:1665^0.393@0.950 W:1754^0.410@0.956 SH:1647^0.311@0.967 H:1781^0.311@0.997 SH:2002@0.964^0.325 H:1727^0.278@0.998 SH:1941^0.278@0.978 SH:1696^0.242@0.986 SH:2023^0.242@0.965 SH:1793^0.266@0.969 W:1867^0.275@0.965
horse	(H:2267@0.725 -H:776@0.996)^0.663 (SH:2201@0.966 -SH:1873@0.977)^0.614 W:2165^0.554@0.927 W:2281^0.513@0.950 W:2153^0.526@0.950 H:2348^0.478@0.994 W:2259^0.481@0.944 H:2357^0.475@0.979 W:2071^0.424@0.950 W:2304^0.451@0.902 RGB:2136^0.368@0.998 SH:2073^0.379@0.970 (H:2256@0.971 -RGB:1805@0.998 -SH:1697@0.988)^0.366 H:2360^0.353@0.976 W:2115^0.278@0.961 H:2181^0.278@0.998 H:2223^0.318@0.986 W:2155^0.266@0.955 W:2082^0.242@0.950 H:2221^0.242@0.997
pear	W:2770^0.995@0.952 W:2860^0.583@0.958 W:2465^0.544@0.957
tomato	H:2882^0.997@0.955 RGB:3178^0.583@0.961

Glossary

boost Factor for a clause or sub query to increase or lower its weight in a boolean query. 130, 132, 142, 177

Caltech-101 The Caltech-101 image repository contains 9144 images with 101 categories. Each category holds 31 to 800 representative images. 58, 109, 111–114

clause Single component (term or sub query) of a boolean query that is either tagged as SHOULD, MUST or MUST_NOT. 60, 69–72, 79, 80, 82, 85, 89, 92, 123, 124, 127, 130, 132, 135–137, 142, 145, 149

ETH-80 The ETH-80 image repository contains 3280 images. The images show 10 representative objects for each of the 8 categories. It is known as the TU Darmstadt Database (formerly the ETHZ Database) [67, 68]. 58, 109, 111–113, 138, 177

F-Measure The F-Measure combines precision and recall into a single value. It is defined by van Rijsbergen [136] as: $F_\beta = (1 + \beta^2) * \frac{\pi * \rho}{\beta^2 * \pi + \rho}$. 58

feature vector A feature vector is a set of data, describing a certain image feature. This data is highly condensed and comparable to other feature vectors of the same kind. Normally, it does not contain enough information to reconstruct the original image. 11, 13–15, 29, 31, 35, 38–43, 45, 46, 54, 55, 57, 62, 64, 69–72, 76, 82, 84, 90, 96, 98, 99, 102–105, 108, 109, 111, 113–116, 118–124, 135, 137, 138, 142–144, 151–155, 175, 176

Histogram Feature vector "Histogram", using 12 stochastic RGB histogram moments for mean, variance, skwness and colour correlation. The identifier within the software is "fv_stochastic2". 100, 102–105, 109, 111, 112, 143, 180

precision Fraction of retrieved documents that are relevant to the search: $\pi = \frac{tp}{tp+fp}$ [136]. 11, 57, 58, 62, 65–67, 69, 70, 72, 82, 98, 116, 123, 124, 127, 130–132, 179

recall Fraction of the documents that are relevant to the query that are successfully retrieved: $\rho = \frac{tp}{tp+fn}$ [136] . 11, 57, 58, 62, 65, 66, 69, 72, 82, 98, 115, 116, 121, 123, 124, 127, 128, 130–132, 134, 136, 179

RGB Mean Feature vector "RGB-Mean", using the average red green and blue values. The identifier within the software is "fv_mean". 100, 102–105, 109, 111–113, 143

semantic gap The term "semantic gap" stands for a core problem of CBIR. It is caused by the difficulty for machines to map visual content (e.g. from pixel images) to a semantic concept. 11–14, 16, 45, 61

slack Additional fraction by which the strict thresholds of clauses are lowered to increase the result set size. This usually increases recall on the expense of precision. 132

Spatial Histogram Feature vector "Spatial Histogram", using a 3 level quad tree divided image, where each sub image is represented by the same 12 moments as in *Histogram*. The identifier within the software is "fv_stoch_quad". 100, 102–105, 109, 111–113

thumbnail A thumbnail image is a downscaled version of an image. It is often used to preview many images on a limited display area. 20, 21, 41

Wavelet Feature vector "Wavelet", using the RGB average and especially the 80 highest coefficients from a Haar wavelet transformation. The identifier within the software is "fv_wavelet". 100, 102, 103, 105, 106, 109, 111–113

Acronyms

ANN Artificial Neural Network. 37, 155

BNF Backus-Naur Form. 23

CBIR Content-Based Image Retrieval. 11–14, 16–18, 23, 24, 26–29, 31–34, 38, 41–46, 48, 50, 52–54, 59, 61, 66, 70, 74–76, 89, 114, 115, 120, 123, 136, 137, 142, 143, 152, 155, 156, 180

CLIR Cross-Language Information Retrieval. 33

CQL Classification Query Language. 27

DBMS Database Management System. 21, 27

DT Decision Tree. 37, 98, 137, 155

EBNF Extended Backus-Naur Form. 39

ECW Enhanced Compression Wavelet. 21–23

Exif Exchangeable image file format. 38

FOQL Fuzzy Object Query Language. 24, 26, 42

GIF Graphics Interchange Format. 22

GIFT GNU Image Finding Tool. 34

GIS Geographic Information System. 12, 27

H Histogram. 103–107, 122

IPTC International Press Telecommunications Council - Information Interchange Model (IIM). 32, 38, 96

JPEG Joint Photographic Experts Group. 21–23

JPEG 2000 Joint Photographic Experts Group 2000. 21–23

KLD Kullback-Leibler divergence. 30, 31

LGPL GNU Lesser General Public License. 22, 23

MDBMS Multimedia Database Management System. 27

MIR Multimedia Information Retrieval. 12, 13, 16, 17, 41

MRML Multimedia Retrieval Markup Language. 37, 85

MrSID Multiresolution Seamless Image Database. 21–23

ODMG Object Data Management Group. 24

OQL Object Query Language. 24

OQUEL Ontological Query Language. 24–26, 42

PGF Progressive Graphics File. 21–23

PNG Portable Network Graphics. 21–23

PR Precision-Recall. 33

QBE Query-By-Example. 14, 26, 52, 53, 80, 85, 89, 98, 99, 102, 137, 152

QBF Query-By-Feature. 52, 85

QBS Query-By-Sketch. 52, 85

RGB RGB Mean. 103–107, 122

ROC Receiver Operating Characteristic. 33

SEQUEL Structured English Query Language. 24

SH Spatial Histogram. 103–107, 109, 123

SOFM Self Organizing Feature Map. 37

SQL Structured Query Language. 24, 27

SVM Support Vector Machine. 37, 152, 155

URI Uniform Resource Identifier. 38, 40

URL Uniform Resource Locator. 26, 52, 79, 81, 89

VIR Visual Information Retrieval. 13

W Wavelet. 103–107, 109, 123

W3C World Wide Web Consortium. 24

XML Extensible Markup Language. 24, 37, 85, 88

XMP Extensible Metadata Platform. 32, 38

Copyright Statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns any copyright in it (the "Copyright") and s/he has given The University of Huddersfield the right to use such Copyright for any administrative, promotional, educational and/or teaching purposes.

ii Copies of this thesis, either in full or in extracts, may be made only in accordance with the regulations of the University Library. Details of these regulations may be obtained from the Librarian. This page must form part of any such copies made.

iii The ownership of any patents, designs, trade marks and any and all other intellectual property rights except for the Copyright (the "Intellectual Property Rights") and any reproductions of copyright works, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property Rights and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property Rights and/or Reproductions.