Applications and Theory of Large Scale Image Retrieval and Large Scale Mining

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by
Stephan Gammeter
Dipl. Phys. ETH
Born April 18, 1983
citizen of Lützelflüh, Switzerland

accepted on the recommendation of
Prof. Dr. Luc Van Gool, examiner
Prof. Dr. Bastian Leibe, co-examiner
Dr. Hervé Jégou, co-examiner

2013
To my wife Azusa, my son Takumi, my brother Christoph and my parents Sepp and Susanne.
Abstract

The state-of-the art in visual object retrieval from large databases allows for searching millions of images on the object level. Complementary works have proposed systems to crawl large object databases from community photo collections on the Internet in order to generate large labeled object datatbases. The dissertation presents applications that are made possible from the combination of said methods and investigates novel ways in which state-of-the art in image retrieval can be advanced. There are four main contributions.

As a first contribution we combine image retrieval with large scale mining methods to create an auto-annotation system for holiday snaps. The resulting method allows for automatic labeling of objects such as landmark buildings, scenes, pieces of art etc. at the object level in a fully automatic manner. We demonstrate the scalability and precision of the proposed method by conducting experiments on millions of images downloaded from community photo collections on the Internet.

As second contribution we present a method to automatically improve the quality of a reference database, which, as we will show, significantly affects recognition performance.

The third contribution presents a system for mobile augmented reality based visual recognition using a hybrid client-server approach. The capabilities of the system are demonstrated with a prototype application on the Android platform, which is able to augment both stationary (landmarks) and non stationary (media covers) objects.

The fourth contribution introduces method based on k-reciprocal nearest neighbors to improve image retrieval. The method operates on the bag-of-words level alone. While it involves offline processing, it only adds minimal computational overhead at runtime. The approach is evaluated on common image retrieval benchmarks and a significant improvement over standard bag-of-words methods is shown.
Zusammenfassung


Als zweiten Beitrag stellen wir eine Methode zur automatischen Verbesserung der Referenzdatenbank vor, welche wie wir zeigen, die erreichte Kennzeichnungspräzision erheblich verbessert.


Acknowledgements

I will never forget the wonderful years of my PhD. I owe this wonderful time to a great number of individuals to all of whom I am truly grateful. My thesis advisor Prof. Dr. Luc van Gool gave me the freedom to pursue the Topics I was most interested in, while sparing no effort in supporting my research. Even before I started my studies at ETH I was interested in computer vision without even knowing it, as I wanted to make computers to be able to see like humans do. If it were not for Dr. Till Quack approaching me after my studies, I would have never found the way back to doing computer vision. And during my PhD he has been a great mentor and friend. During the very early period of my PhD, Prof. Dr. Bastian Leibe helped me to hit the ground running. With his can do attitude and immense support in authoring my first publication. My excellent colleagues Lukas Bossard, Andy Ess, Danfeng Qin and Matthias Dantone were all instrumental in keeping the morale high during the rough patches before deadlines and made me go to work every day with a smile on my face. Outside of the Institute I received immeasurable support and love from my wife Azusa, my brother Christoph and parents Sepp and Susanne. To all of you, I cannot say it enough: Thank you!
## Contents

1 Introduction ........................................ 1  
   1.1 Contributions of this thesis .................. 1  
   1.2 Organization of this thesis .................. 3  

2 Visual Word based Large Scale Image Retrieval ... 5  
   2.1 Local Invariant Features ...................... 7  
   2.2 Quantization and Indexing ..................... 10  
   2.3 Feature space learning ....................... 12  
   2.4 Geometric Verification ....................... 12  
   2.5 Query Expansion ................................ 14  
   2.6 Correlated Features ......................... 14  
   2.7 Evaluation Methodology ....................... 15  

3 Large Scale Autoannotation .......................... 17  
   3.1 Introduction .................................. 17  
   3.2 Related Work .................................. 19  
   3.3 Automatic object mining ...................... 21  
   3.4 Scalable object cluster retrieval .......... 23  
   3.5 Object knowledge from the wisdom of crowds .. 24  
      3.5.1 Object-specific feature confidence score ... 25  
      3.5.2 Better indices through object-specific feature sampling .................. 26  
   3.6 Object-level auto-annotation ................. 27  
   3.7 Experiments and Results ...................... 28  
      3.7.1 Efficiency and Precision of Recognition ... 30  
      3.7.2 Annotation precision ..................... 32  
   3.8 Conclusions .................................. 34  

4 Improving Large Scale Autoannotation ............... 37
4.1 Introduction ................................................. 37
4.2 Mining and Recognition of Objects ................. 39
  4.2.1 Object Mining ........................................ 39
  4.2.2 Object Recognition ................................. 41
4.3 Cross-Media Cluster Analysis ....................... 42
  4.3.1 Expansion of small clusters ...................... 42
  4.3.2 Efficient Itemset Query Selection ............... 44
  4.3.3 Reduction of large clusters ..................... 46
4.4 Experiments and Results ........................... 48
  4.4.1 Object Recognition ................................. 49
  4.4.2 Cluster reduction statistics ...................... 50
  4.4.3 Efficient Itemset Query Selection ............... 50
  4.4.4 3D Reconstruction ................................. 52
4.5 Conclusion .................................................. 54

5 Mobile Visual Search ................................. 55
  5.1 Introduction ............................................. 55
  5.2 System overview ....................................... 58
  5.3 Server-side object recognition .................... 60
    5.3.1 Landmark recognition ............................. 60
  5.4 Client-side object tracking ....................... 63
    5.4.1 Visual feature tracking ......................... 64
    5.4.2 Motion estimation ................................ 65
    5.4.3 Sensor tracking ................................... 67
  5.5 Implementation details ............................. 70
  5.6 Experiments and results ......................... 70
  5.7 Conclusions ............................................. 71

6 Reciprocal Nearest Neighbours ..................... 73
  6.1 Introduction ............................................. 73
  6.2 Related work .......................................... 74
  6.3 Motivation .............................................. 76
  6.4 Our Approach .......................................... 80
    6.4.1 Close set construction ........................... 80
    6.4.2 Far set re-ranking ................................ 83
  6.5 Experiments ............................................. 84
    6.5.1 Evaluated datasets ............................... 84
    6.5.2 Close set accuracy ............................... 85
6.5.3 Far set accuracy ........................................... 86
6.5.4 Full method ................................................. 86
6.6 Close Set reranking examples ................................. 92
6.7 Efficient representation of the neighbourhood graph ... 98
6.8 Conclusion ....................................................... 100

7 Engineering Aspects ............................................. 101
  7.1 Features ........................................................ 101
     7.1.1 Binaries .................................................. 101
  7.2 Feature Scaling ............................................... 106
  7.3 Document similarity function ............................... 107
  7.4 Conclusion ..................................................... 112

8 Conclusions and Outlook ...................................... 113
  8.1 Contributions ............................................... 113
  8.2 Outlook and open questions ................................. 115

Bibliography ....................................................... 117
Introduction

Vision is arguably the most important tool we have to navigate the world. Understanding what we see comes so naturally to us that it is often difficult to explain to someone outside of the computer vision community why computer vision is so hard. After all it is the easiest thing in the world to tell apples and oranges apart. However if we are not supplied with an actual image but rather a large text file of bitmap values, thereby circumventing a big part of the image processing wetware of our brain, we can easily see why computer vision is so hard.

Nevertheless astounding progress has been made in Computer Vision in the last decade. While there are still core problems which remain largely unsolved — and in cases where progress has been made, we don’t yet fully understand why things exactly work as well as they do — especially in Image Retrieval we have seen amazing advances. The amount of images that Image Retrieval Systems can handle has grown by several orders of magnitude. Starting at proof of concept systems with only a few thousand images, Image Retrieval Systems have now been commercialized and handle billions of images.

1.1 Contributions of this thesis

This dissertation is about applications of Image Retrieval and various ways in which Image Retrieval can be improved. Some of the improvements shown are specific to an application, while others work in very general setting. All in all there are four main contributions.

As first contribution an Image Retrieval based auto-annotation system is presented. The system is used to automatically annotate holiday pictures supplied
by users. The annotation consists of an estimated geolocation as well as a short text description. The system presented is a fully automated pipeline. In a first step, geotagged images from community photo collections are collected in order to build a reference database of landmarks. The mined photos are then indexed and used to automatically tag photos uploaded by users. The entire process from collecting photos to annotating new unseen pictures is automatic and does not require any human interaction.

As second contribution, a method for improving the auto-annotation systems reference database is presented. Originally only geotagged photos are used to identify landmarks. Using the associated tag information of the initial geotagged images, text queries are constructed that are used to search for new images via Internet search engines. Since only a fraction of images on the Internet are geotagged, this enables the system to make use of a much larger set of images. Additionally a simple method is introduced to purge redundant images from the reference database with virtually no loss in accuracy.

As third contribution the auto-annotation system is integrated with mobile devices to give the user a see through augmented reality user interface. The mobile device acts as a client to the auto-annotation system. It periodically (∼ once per second) queries the annotation system with the current frame recorded by the camera. Since the request round trip time is much too large and query frequency is much too low to meet the update speed required by an augmented reality application, a client side tracker is used. Furthermore improvements on the server side are shown to increase the processing speed of incoming queries.

The fourth contribution is a method for improving image retrieval precision and recall with virtually no computational overhead at runtime. The method is based on the notion of reciprocal nearest neighbours in document space. Traditionally Image Retrieval is seen as a nearest neighbour problem. Querying the system boils down to finding the $k$-nearest neighbours of the query image amongst the database images. This approach however implicitly assumes that the document space is homogeneous in the sense that distances in one region of the space can be directly compared to distances in other regions of the space. Unfortunately this is not the case. Considering $k$-reciprocal nearest neighbours instead of only $k$-nearest neighbours helps deal with this inhomogeneity, since not only the $k$-neighbourhood of the query is considered, but also all $k$-neighbourhoods of the $k$ query neighbours.
1.2 Organization of this thesis

Chapter 2 gives a short introduction to Large Scale Image Retrieval. Over the last couple of years visual word based retrieval has crystallized itself as the most commonly used architecture. This chapter gives an overview of all components of visual word based retrieval. In the context of each component related work is discussed. Finally this chapter ends with a look at common evaluation methodologies.

Chapter 3 presents an Image Retrieval based autoannotation system for holiday pictures. The system is fully automated in that it automatically mines geotagged images in community photo collections for landmarks to build up a reference database. The database is then indexed and used to annotate user submitted images. Recognized images are tagged with a text describing the landmark they show and if available a matching Wikipedia article for further information. Additionally a geotag is also provided to the user.

Chapter 4 demonstrates an improvement on the autoannotation system shown in Chapter 3. The original system is limited in the sense that it requires geotagged images to build up the reference database. This issue is addressed by automatically extracting keywords which are then used as text queries to query Internet search engines for additional images. It is demonstrated that adding these additional images significantly improves retrieval performance. Furthermore a simple graph based method is used to purge uninformative images from the database with virtually no loss in retrieval performance.

Chapter 5 integrates the work of Chapter 3 and 4 into a mobile augmented reality system for landmark recognition. The client server architecture is discussed as well as the object tracking algorithm which runs in real time on the mobile client. Additionally some speed improvements are made on the system presented in the previous chapter, so that it delivers responses fast enough for augmented reality.

Chapter 6 discusses the reciprocal nearest neighbours based method for improving retrieval in a very general setting. The method is evaluated on several popular datasets and in each case a significant improvement is observed. While
this method is very robust and effective, it adds non negligible storage overhead. This is partly addressed using simple compression methods that greatly reduce the required storage.

**Chapter 7** evaluates several feature detectors, feature descriptors and similarity functions in detail using the Oxford5k, Paris, Kentucky and Holidays datasets. It is demonstrated, that the choice of the local feature detector and its subsequent descriptor have a significant effect on retrieval performance. Additionally it is shown, that small modifications made to the feature descriptor and ranking function significantly improves retrieval accuracy.

**Chapter 8** concludes the thesis and discusses promising directions for further research.
Visual Word based Large Scale Image Retrieval

The goal of Image Retrieval is, given a set of database images and a query image, to find all images in the database that are “similar” to the query image. Where “similar” refers to the human perception of similarity. Since the notion of visual similarity has a very broad meaning, the human perception of similarity is notoriously difficult to capture. Two images can be similar because they feature the same painting. Alice finds that her cat looks similar to Bob’s cat, while Mallory thinks that cats in general are similar to tigers. If we think of similarity of two images as a binary function, which decides whether or not a given image pair is similar, then these different notions of similarity can be seen as allowing different numbers of degrees of freedom. Two images of the same painting, for instance, only differ by the viewing angle and potentially some lighting differences, while two images of the same cat additionally vary by the articulation of the cat, which adds a tremendous amount of degrees of freedom. State of the art Image Retrieval currently only deals with the former situation where there are only a few degrees of freedom. This encompasses mostly static and textured object like buildings, landmarks, paintings etc. Despite this restriction, Image Retrieval has a plethora of applications some of which go way beyond the scope of this thesis.

Since it was originally proposed by [Sivic and Zisserman 2003], the Visual Word based approach to Image Retrieval has been a shining success story in Computer Vision. The approach is so simple and easy to understand, that it can be explained within only a few minutes to someone unfamiliar with Computer Vision. Despite its simplicity it has demonstrated excellent performance in precision and recall as well as excellent scalability.
Figure 2.1 gives an overview of all components necessary for visual word based retrieval. In a first step for every image, local invariant features are extracted (e.g. [Lowe 1999] [Matas et al. 2002] [Bay et al. 2006]). Usually there are around 1000 local features per image and depending on what type of feature used, each feature is typically represented as a 64 or 128-dimensional vector. All local invariant feature vectors are then quantized, which is usually done by a clustering algorithm like $k$-means, where $k$ is usually somewhere in the range of $10^5$ to $10^6$. There are many different quantization schemes. However, a common denominator is that the number of quantization bins by far exceeds the number of local invariant features per images (1000 per image). After quantization, each image is represented as a large sparse vector, where each entry represents a quantization bin and counts how many times a local invariant feature was quantized to said bin. The quantized local features are usually referred to as “visual words”, since every quantization bin corresponds to a certain type of image patch. The “visual word” vectors are also commonly referred to as “bag of feature-vectors” (bof-vectors) or “bag of word-vectors” (bow-vectors). A similarity or distance function is usually defined in terms of bow-vectors, popular choices are the intersection-over-union measure as well as the cosine similarity measure. The sparsity of the bow-vectors can be exploited to enable very fast computation of the similarity between the query
2.1. Local Invariant Features

Arguably one of the most influential ideas, Local Invariant Features are now encountered in almost every computer vision application. The main idea is to step away from a global view and instead focus only on recognizable patches of an image. This approach has many advantages, one of which for instance is robustness to clutter and partial occlusions. Furthermore, many global transformations can locally be approximated by much simpler transformations, for instance a projective transformation (8-dof) can be locally approximated by an affine transformation (6-dof). In Figure 2.2 several examples of local features are shown. The extraction of Local Invariant Features is usually a two-stage process.

In the first step (region detection) recognizable regions of an image such as blobs or corners are detected. Then for every region an invariant frame is computed. For instance in [Lowe 1999] the maxima and minima of a difference-of-Gaussian filter are used to find blob like structures. Note here that a difference-of-Gaussians filter can well approximate a Laplace-of-Gaussian filter, and since the Laplace operator is self-adjoint this basically means that maxima and minima of the curvature function on a Gaussian-smoothed version of the image are found. The width of the Gaussian determines at what scale the curvature function is computed. The invariant frame in [Lowe 1999] besides it’s center has only two additional degrees of freedom, scale and orientation. The scale is determined by a local maximum in scale space and the orientation is determined by binning local gradient of the image patch and picking the bin with the highest response.

In the second step (feature description), a feature descriptor is computed on the invariant frame. A very popular choice for the feature descriptor is a histogram of oriented gradients. The are of course many variants, but generally the idea is to partition the local invariant frame into several bins and to compute for each bin a orientation histogram of gradient responses. This approach has some very nice properties. For one, since a local invariant frame is partitioned into multiple bins, a lot of information is kept on the local arrangement of
gradients. More importantly, however, this approach is invariant to any monotonic increasing transformation of the image intensity, since the local gradient direction does not change with such transformations.

The number of proposed local invariant features is ever growing and discussing them all would go well beyond the scope of this thesis. Probably the most frequently used local feature descriptor is SIFT [Lowe 1999]. SURF [Bay et al. 2006], another very popular region detector / feature descriptor pair, uses Haar wavelets to approximate derivatives and uses the determinant of the Hessian matrix instead of the trace \( i.e. \) the Laplace operator) as a blob detector. Since the response of haar wavelet filters can be efficiently calculated using integral
images, SURF features can be computed several times faster than SIFT features. The original SIFT [Lowe 1999] [Lowe 2004] region detectors are only scale and rotation invariant. In [Mikolajczyk and Schmid 2004] the authors use an iterative procedure to compute affine invariant frames around interest points. Another very popular region detector is MSER [Matas et al. 2002], here maximally stable extremal regions are used as feature points, which are even invariant under projective transformations of the image. ASIFT [Yu and Morel 2011] takes a somewhat different approach. Instead of computing affine invariant frames on the input image directly, simple SIFT features are extracted from several transformed copies of the original image. This approach allows for finding image correspondences even in very extreme cases where Hessian Affine [Mikolajczyk and Schmid 2004] or MSER blob detectors with SIFT as a feature descriptor fail. While the previously mentioned region detectors focus on accuracy rather than runtime, FAST [Rosten and Drummond 2006] is a very fast corner detector which is often used for real time applications and on embedded devices. On a side note, one should mention that in many applications of computer vision, the region detection step is replaced by a dense sampling of the entire images, however this is quite uncommon in the realm of “Large Scale Image Retrieval”.

While there is not only a plethora of region detectors, there is probably an even larger number of feature descriptors. Many of those, are in some way or another a histogram of oriented gradients and differ mostly in the way gradients bins are positioned inside an invariant frame. For instance, whereas SIFT and SURF use a rectangular grid of bins, GLOH [Mikolajczyk and Schmid 2005] uses a log-polar grid of bins for the orientation histograms. While the SIFT descriptor places each gradient pixel into multiple histogram bins using a weighted sum (in order to avoid boundary effects due to histogram boundaries of the SIFT descriptor), Daisy [Tola et al. 2008] uses a sum of convolutions to avoid boundary effects, which can be computed several times faster. An extension of DAISY is presented in [Winder et al. 2009]. The authors use a ground truth dataset of correspondences to learn the optimal binning configuration and to find the best configuration of low-level filters.

For the most part, this thesis only considers SIFT [Lowe 1999] and SURF [Bay et al. 2006] features, as they are considered to have the best performance for “Large Scale Image Retrieval”.

2.1. **Local Invariant Features**

---

*Lowe 1999*

*Lowe 2004*

*Mikolajczyk and Schmid 2004*

*Matas et al. 2002*

*Yu and Morel 2011*

*Rosten and Drummond 2006*

*Mikolajczyk and Schmid 2005*

*Tola et al. 2008*

*Winder et al. 2009*
2.2 Quantization and Indexing

The storage required to represent extracted features of an image is usually on the same order of magnitude as the image file size itself. For instance, if 1000 64-dimensional SURF-features are encoded using 32-bit floats, then this amounts to 250 kB, which is about the same amount of storage, required for a one-megapixel JPEG encoded image. Quantizing feature descriptors has two main advantages. For one the storage requirements can be significantly reduced. Another advantage is that assuming an image distance measure is formulated directly in terms of quantized features, then the distances of a query image to all database images can often be efficiently computed using an inverted index data structure. Of course this strongly depends on how the distance measure is defined and does not hold in general. Basically the distance measure must be defined in a way such that features which fall into different quantization bins only affect the calculated distance in a trivial manner. This can be illustrated using a simple example. Consider the following situation: we have a database with one million images, where each image contains one thousand feature points. Each feature is quantized using 20 bits (i.e. $2^{20}$ visual words) for instance using $k$-means with $k = 2^{20} \approx 1'000'000$ centroids. The similarity measure is simply the number of visual words that two images share. In this situation, only $20'000$ bits are required for representing each image amounting to about 2.3 GB of total storage, as compared to roughly 230 GB required to store unquantized feature vectors. Furthermore using an inverted index, only a fraction of these 2.3 GB has to be processed for a given query image. An inverted index is a map from the visual words to the set of documents that contain said visual word. The set of documents for a visual word is usually referred to as “posting list”. Assuming all posting lists are of equal length, then for a query image with 1000 visual words only $1'000/1'000'000 = 0.1\%$ of all posting lists have to be processed in order to calculate all distances from the query to each database image. This corresponds to only 2.3 MB of data for this example.

Many important aspects were of course ignored here. For instance, doing $k$-means with $1'000'000$ centroids is no trivial task, and there is no mention on how the information loss caused by the quantization affects retrieval accuracy. The combination of quantized local features with an inverted index first appeared in [Sivic and Zisserman 2003]. The authors used regular $k$-means to train vocabularies with $k \approx 6'000 - 10'0000$ depending on what type of local
feature was used. Nevertheless they achieved some very impressive results at that time in a video retrieval scenario. Later [Nistér and Stewénius 2006] used hierarchical $k$-means to train much larger vocabularies. The authors found that with a bigger vocabulary (*i.e.* more bits spent per feature) the recognition performance increases. They considered vocabularies with up to 16 million visual words and demonstrated results on a database of 40'000 images. They found that, due to the hierarchical approach quantization artifacts were introduced, which they alleviated by using a hierarchical scoring scheme. One year after that [Philbin et al. 2007] used a best-bin-first modified kd-tree [Beis and Lowe 1997] for approximate nearest neighbour search. The kd-tree was used as a fast approximation to the nearest neighbour search of regular $k$-means. With this modification the authors were able to cluster large visual vocabularies with up to 1.25 million visual words in reasonable time. They also found that with a larger vocabulary size retrieval performance increases. This held up to 1 million visual words, after that a slight decrease in performance was observed. Their method clearly outperformed the hierarchical $k$-means method of [Nistér and Stewénius 2006]. One can clearly see a pattern here, with better quantization, *i.e.* more accurate representation of the original features, comes better performance. And indeed in the following years several papers improved on previous results by reducing quantization effects. Building on [Philbin et al. 2007] the authors of [Philbin et al. 2008] assigned each feature to multiple visual words, thereby increasing the number of bits used to encode each feature. While this increases the computational cost of a query, they found that this quantization scheme was not only superior to the single assignment but also works better when the visual vocabulary is trained on an independent image dataset. Moving away from best-bin-first kd-trees [Jegou et al. 2008] used a relatively small vocabulary of $20k$ visual words which is trained using exact $k$-means or an average sized vocabulary of $200k$ trained via with $k$-means in conjunction with binary signatures for each feature. The goal of the binary signature is to approximately localize features inside the Voronoi cells created by the vocabulary. They are generated by hyperplanes passing through each Voronoi cell and each bit indicates on which side of a hyperplane a feature lies. The advantage of this approach is that it is very fast even when a small vocabulary is used, since the comparison of binary signatures is extremely simple. Furthermore smaller vocabularies lead to increased recall of potentially matching features as compared to large vocabularies, where the Voronoi cells are much smaller and often matching features can end up in different cells.
2.3 Feature space learning

Up to this point, only methods were discussed which quantize features better. However it is somewhat naïve to assume that distances in the SIFT or SURF feature space correctly reflect visual similarity. Furthermore these feature spaces are very non-uniform. If for instance the region detector detects both bright-on-dark blobs and dark-on-bright blobs, then usually two non-connected components can be found in the feature space, where one corresponds to dark-on-bright blobs and the other to bright-on-dark ones. Of course simply the usage of a region detector already limits the region in feature space where features can be placed, since all features correspond to blob like structures. However the biggest issue is that distances between features from similar image patches vary depending on the image patch. Therefore any quantization scheme which does not account for this will be to coarse in some regions of the feature space and too fine in other regions. Getting rid of this anisotropy is therefore desirable, and can potentially be done with training data generated in an unsupervised manner using wide baseline stereo matching methods. At the same time both [Mikulík et al. 2010] and [Philbin et al. 2010] implemented this idea. Both papers are quite similar and they both use wide baseline stereo-matching methods for generating training data. However they handle the aforementioned anisotropy differently. The authors of [Mikulík et al. 2010] used a very fine vocabulary with 16 million visual words and learned which Voronoi cells correspond to an actual instance of an image patch. Whereas [Philbin et al. 2010] used deep belief networks to learn an embedding of the original descriptor space in a lower dimensional space where the Euclidean distance more accurately reflects visual similarity.

2.4 Geometric Verification

The arrangement of local features can play a vital role in determining if two images show the same object or not. In the bag-of-words approach, where each image is only represented by a vector of occurring visual words, this information is simply discarded. While somewhat surprisingly approaches ignoring this information still work quite well, geometric information can help remove false positive images that just happen to have similar features to the ones in the query image. Several methods have been developed in recent years that address this opportunity. In [Sivic and Zisserman 2003] a simple approach was
taken. The authors loosely checked geometry by verifying that neighbouring features in the query image match to neighbouring features in the retrieved image. A more strict method to ensure geometric consistency is to use RANSAC (or some variant thereof) to estimate image transformations from the query image to the retrieved image. The correspondences required by RANSAC can be directly taken from the matching visual words. However, since this is still computationally expensive, it can only be done for a subset of all retrieved candidate images. In [Lowe 2004] a similar idea was used, but instead of RANSAC they first use a Hough voting scheme to find groups of features that share the same scale, rotation and translation parameters (4 dof). Then based on these feature clusters an affine transform is estimated (6 dof). The authors of [Philbin et al. 2007] took advantage of the affine invariant frames calculated for each feature and used single feature matches to estimate an image transformation (they restricted possible transformations to affine sub-groups). As only single feature matches are considered, there are only a couple of hundred image transformation hypothesis which need to be tested. Therefore this can be done exhaustively. This approach is however still not fast enough to run on all retrieved candidate images and so this step is only applied to the top ranking candidates.

While the aforementioned approaches do not directly modify the inverted index approach itself, but rather just are an additional post-processing step, [Jegou et al. 2008] put geometric information directly into the inverted index. For each feature they additionally store discrete approximations of the log-scale and rotation in the inverted file. At query time independent log-scale and rotation histograms are populated with rotation and log-scale differences from the query and database features. Then all features are discarded, except the ones in the minimum bin of the maximal scale bin and maximal rotation bin. While this method only considers rotation and scale it can be applied to all candidate images unlike the previously mentioned methods.

In [Perdoch et al. 2009] the authors demonstrate a method to quantize the shape of local affine frames thereby reducing the amount of storage required. They however do not use this information to perform global geometric verification like [Jegou et al. 2008], but rather use it to efficiently perform geometric verification on the top ranking images as in [Philbin et al. 2007].
2.5 Query Expansion

Image retrieval can be seen as a nearest neighbour search problem, where the retrieved images are the nearest neighbours. However this definition requires the distance function to truly reflect visual similarity which is often not the case. Query expansion offers a mechanism to implicitly modify the similarity function for each query, such that it better lines up with the human notion of similarity. The principal idea is to first query the database for similar images and then filter out candidate images which are highly likely to correspond to the query image. In terms of document space vectors, one looks for a neighbourhood in which images are very likely to correspond to the query vector. This neighbourhood is then used as a query against the rest of the database images and the distance function is therefore modified in the sense that it becomes dependant on the local neighbourhood of the query. The authors of [Chum et al. 2007] evaluated several methods to do this. The methods range from a simple approach, where the mean of the 5-neighbourhood around the query is used to build a new query vector, to more sophisticated approaches where larger neighbourhoods are filtered using geometric verification and individual visual words are backprojected to the query image, in order to determine if they should contribute to the newly constructed query vector. Traditionally, query expansion depends on a mechanism that reliably finds additional images that correspond to the query image. This is, however not always possible in visual word based image retrieval, as correlated features or bursts which are discussed in the following section can lead to many unrelated images occurring in the toplist of a given query. In such a scenario geometric verification might simply fail on large parts of the toplist and only few additional images can be found for query expansion. This problem is addressed in [Chum et al. 2011]. Their system detects the aforementioned situation (they refer to it as tf-idf failure) and then from the knowledge that most images are not relevant to the query image a model for the correlated features is built in order to suppress their effect. This can be seen as the opposite of query expansion, as features are effectively removed from the query.

2.6 Correlated Features

Most similarity or distance functions used in visual word based image retrieval implicitly assume that individual visual words occur independently. Unfor-
2.7. Evaluation Methodology

Unfortunately this is however far from true. Consider for instance a window of a building, this window will usually be covered with multiple local features. So whenever the such a window appears in an image all those local features will appear as well, therefore they do not occur independently. Given that once such feature appears in an image, then the chance that the other occur as well is also high. This situation can arise in many cases, regular structures like fences, grids or building facades but also many textures like water or trees cause cooccurring features. In the literature these features are usually referred to as bursts [Jégou et al. 2009], cocsets [Chum and Matas 2010] or confusers [Knopp et al. 2010]. There are many ways in dealing with this problem. In Cummins and Newman 2008 the authors use a Chow Liu tree [Chow et al. 1968] to approximate the joint distribution of the visual words, therefore exploiting the information given by cooccurring features. Contrary to this, other approaches identify cooccurring features and try to minimize their negative effect on the retrieval process. Different scoring schemes (i.e. similarity functions) are evaluated in [Jégou et al. 2009] which aim at reducing the effect of bursty visual elements. An offline approach is presented in [Chum and Matas 2010], they use min-hash to efficiently identify correlated features and then either remove them completely or use an adapted similarity function not unlike [Jégou et al. 2009] to reduce the negative effect that correlated features cause.

2.7 Evaluation Methodology

The most popular measure to evaluate a retrieval system is “mean average precision”. Given a database of images, query images and an ground truth data which assigns database images to query images, “mean average precision” or \( mAP \) is computed by taking the mean of the average precision of each query. The average precision is simply the area under the precision recall curve created for each ranked list. One reason why \( mAP \) is so popular, is that it is very intuitive to understand and measures precision as well as recall in a natural manner. However there are some issues with \( mAP \). For instance it gives a heavy weight to the first few entries of ranking lists. Also in some scenarios \( mAP \) does not correctly reflect how useful a system is. One such scenario is given in Chapter 3 for instance, where image retrieval is used to solve a recognition task. In this case it is of no interest to the user that the result he is looking for has a high chance of appearing in the top 10 ranked images, rather he wants to know that single highest weighted image returned by the system is the cor-
rect one. In short, a good $mAP$ implies that the system is doing something right, but bad $mAP$ scores do not necessarily mean that the system is bad.
3

Large Scale Autoannotation

3.1 Introduction

These days, an increasing amount of photos is being stored on desktops and the Web. For instance, the social networking site facebook \(^1\) reports that 30 million photos are uploaded to the site by its users – daily. Most of these photo organization tools also allow for some form of tagging (labeling) with keywords to facilitate search in the photo collection. However, tagging is a tedious process, and the rapidly growing amount of digital photos calls for some form of automated annotation.

In this chapter we present a system which tags photos automatically with keywords referring to places, landmark buildings or events present in a photo. The annotation reaches down to the object level, \(i.e.\) recognized items are outlined with a bounding box in the image. The whole system works without any manual creation of a reference database and can annotate query images with data from millions of reference images in seconds. Figure 3.1 shows a typical example outcome from our annotation system. To achieve this functionality we combine two lines of recent work. First, in order to automatically create a reference database we build on large-scale data crawling from community photo collections [Quack \textit{et al.} 2008]. Second, for recognition from that database we integrate scalable visual vocabulary based recognition approaches [Chum \textit{et al.} 2007, Chum and J.Matas 2008, Jegou \textit{et al.} 2008, Nistér and Stewénius 2006, Philbin \textit{et al.} 2007, Philbin \textit{et al.} 2008, Sivic and Zisserman 2003, G. Schindler and Szeliski 2007].

\(^1\)www.facebook.com
The first step, the crawling stage, enables us to create a large database of (exemplar) object models. Each object is represented as a cluster of images which show the same entity (object, event, scene etc.). In addition, the crawling stage also tells us what the cluster contains, by proposing labels, GPS location, and related content on the Internet without any manual intervention. This information is collected from the meta-data associated with the images from the community photo collections.

The second stage, the retrieval stage, consists of a large scale retrieval system which is based on local image features. It indexes the entire database that was collected in the previous step. We further optimize the retrieval stage by integrating knowledge about the objects into the index creation process. This knowledge is also automatically acquired during the crawling stage. It results in more compact indices (only 33% of the original size) without significant loss in precision. Any query image for annotation is first sent to the retrieval system, in order to identify matching objects. A verification step based on
multiple view geometry refines the hypotheses. Finally, an annotation stage estimates the position of the object within the image, and annotates it with text, location, and related content from the database, resulting in the final annotation as the example shown in Figure 3.1.

In summary, the contributions of this work consist of:
1) The combination large-scale object mining from the Internet with scalable retrieval to obtain an auto-annotation system driven by the wisdom of crowds.
2) The exploitation of knowledge collected during the mining stage to improve object retrieval.
3) The automatic annotation of objects in holiday snaps at the object level - with bounding box, related text, related Internet content and geographic location.

The remainder of this chapter is organized as follows. A discussion of related work follows in the next section. In section 3.3 we discuss how a reference database is mined which is used in the object annotation stage. In section 3.4 we discuss object retrieval and in section 3.5 we introduce several improvements to the retrieval based on the knowledge from the mining stage. In section 3.6 we combine the results from mining and retrieval stages to create a large-scale auto-annotation system.

3.2 Related Work

Our annotation method relates to other works in several aspects. In general, auto-annotation is one of the most relevant applications for object recognition methods, and accordingly a large amount of work has been devoted to that problem [Barnard et al. 2003, Berg et al. 2004, Bosch et al. 2006, Jin et al. 2005, Lavrenko et al. 2003, Schaffalitzky and Zisserman 2002, Torralba et al. 08, Vogel and Schiele 2007, Wang et al. 2006]. Types of annotation cover a wide range, from scene classification [Bosch et al. 2006, Vogel and Schiele 2007, Barnard et al. 2003], over assigning names to faces [Berg et al. 2004], to learning general correlations between words and image content [Barnard et al. 2003, Lavrenko et al. 2003]. While many of the classic auto-annotation works deal with collections such as the Corel database, some of these works also integrate information from Web-image collections [Wang et al. 2006].
Work with photos from community photo collections on the Internet for retrieval of landmarks, locations *etc.* is a topic which is gaining increasing attention recently. For instance, in [Hays and Efros 2008] the geographic location an image was taken at is estimated by comparing it to an enormous database of images downloaded from Flickr. The overall objective is to find near duplicate images of the same scene very efficiently. The images are encoded using several global feature types. The location of the picture is estimated by finding the nearest neighbor(s) in the database. This results in recognition rates of up to 16% for locating an unseen test-image within 200 km of its correct location. However, these recognition rates are not sufficient for the kind of auto-annotation applications we have in mind. A more precise, but more costly processing is possible with local image features. For example, in [Simon et al. 2007, Snavely et al. 2006] a method for clustering images from community photo collections was proposed using multi-view geometry based matching of local features. The goal was to derive canonical views for certain landmarks and to use those as entry points for browsing. Initial image collections for clustering were retrieved by querying photo collections with known keywords such as “Rome”, “Pantheon”, *etc.* In [Quack et al. 2008] the authors proposed an approach, which relies on geo-tagged images, thereby avoiding the need for manual query generation. Both works focus solely on the database creation process. In another notable paper by [Weyand and Leibe 2011], the authors propose an efficient algorithm to find landmarks in large, unstructured image collections, by searching for local attractors in the image distribution that have a maximal mutual homography overlap with the images in their neighborhood.

In this chapter, we combine a data collection process similar to [Hays and Efros 2008, Quack et al. 2008, Simon et al. 2007, Snavely et al. 2006] with object retrieval methods [Chum et al. 2007, Chum and J.Matas 2008, Jegou et al. 2008, Nistér and Stewénius 2006, Philbin et al. 2007, Philbin et al. 2008, Sivic and Zisserman 2003, G. Schindler and Szeliski 2007] to obtain a large-scale, retrieval-based auto annotation system. The resulting method differs from earlier annotation systems in several ways. First, the task we set out to solve is not general annotation of images with words (such as tiger or grass), but rather the labelling of specific objects such as landmark buildings (*e.g.* Eiffel Tower, Teatro di Marcello), as they are often present in typical holiday snaps. Second, the annotation happens at the object level, *i.e.* the object is outlined with a bounding-box in the image. Besides textual labels, the annotation also includes related web-sites describing the object and its GPS position. Third, the
3.3 Automatic object mining

The first element of our annotation system is a large collection of photos for objects (e.g. landmark buildings, statues, mountains, scenery, ...), which will serve as a reference database for later annotation of query images. The key success factor is, to somehow eliminate all the irrelevant information, such as dog pictures, party pictures etc. To collect such a high-quality database automatically, we follow the approach proposed by Quack et al. in [Quack et al. 2008]. Their work describes a system, which crawls community photo collections on the Internet to identify clusters of images referring to a common object or event. The clusters are created based on the images’ pair-wise visual similarities, and the meta-data of the clustered photos is used to derive a labeling for the clusters and to crawl related content on the Internet. The whole process is automated and does not require any manual intervention. The beauty of this method is that it combines the stored collective intelligence of Internet users to distill a reference database of objects. In summary, the system proposed performs the following steps:

1. A geospatial grid is overlaid over the earth. For each grid tile’s center a query is sent to Flickr, to retrieve geo-tagged photos from that area.

2. For each tile, the retrieved photos are matched pair-wise using local visual features and a homography as geometric filter on the features’ positions. The number of inliers after homography verification gives a similarity measure for each pair of photos. The resulting distance matrix is used to cluster photos into groups of images showing the same object or scene. Since this expensive pair-wise matching step is done only per geographic cell (which typically contains significantly fewer than a thousand images in average) it is scalable and can be executed in parallel for each geographic tile. Figure 3.2 shows typical image clusters as they are created in this stage. For sake of simplicity, we will refer to these clusters as “object clusters” in the following (in spite that they might also contain images of scenery, events etc.).
3. The meta-data of each object cluster’s photos is used to label the object clusters automatically. Textual labels are estimated using frequent itemset mining [Agrawal et al. 1993] on the associated text (tags, titles etc.).

4. Possibly related Wikipedia articles are crawled using the text labels from the previous steps as keyword queries in order to search for Wikipedia articles. A final verification is performed by extracting images from each article and trying to match them back to the associated photo cluster. If matching images can be found, the article is assumed to be relevant for the cluster. Note that this final step is essential, to obtain relevant related content for each object cluster.

We implemented the same pipeline as proposed in [Quack et al. 2008], but extended it with some modifications. The most important modification is that we do not rely on a regular geographic grid only to crawl photos from Flickr. Instead, we also instantiate crawling from the locations of geo-tagged Wikipedia articles. We start out a list of all articles on Wikipedia which are annotated
3.4 Scalable object cluster retrieval

In order to search the database of object clusters, we employ state-of-the-art image retrieval methods based on visual vocabulary techniques. Visual vocabularies are usually created by clustering the descriptor vectors of local visual features such as SIFT [Lowe 1999] or SURF [Bay et al. 2006]. This approach has been applied very successfully in the domain of video retrieval [Sivic and Zisserman 2003] using k-means clustering to build vocabularies. Nister et al. demonstrated the benefits of larger vocabularies using a hierarchical k-means variant [Nistér and Stewénius 2006], which allowed retrieval for millions of images. The main advantage of the hierarchical approach is that it allows both for the efficient creation and search of the visual vocabulary. Recently, Philbin et al. showed [Philbin et al. 2007] that a vocabulary obtained from a flat clustering still outperforms the hierarchical approach in precision. To handle large vocabulary sizes without an efficient hierarchical clustering method, they proposed approximate nearest neighbor search to speed-up cluster assignment using a kd-forest as an index on the cluster centroids. (This
is done while clustering as well as for searching the vocabulary with a query point).

Due to its superior performance, we build on the approximate k-means (AKM) idea [Philbin et al. 2007] to index our database of images. We re-implemented the method and achieved similar performance on the Oxford test-set from [Philbin et al. 2007]. (We obtained 0.53 mAP without geometric constraints etc. compared 0.6 mAP in [Philbin et al. 2007] and 0.5 mAP in [Jegou et al. 2008], both also without geometric constraints).

Typically the retrieval results are ranked using a TF*IDF [Manning et al. 2008] scheme and a geometric consistency check for the arrangement of the local features in the image. In our system we use TF*IDF with

\[
TF = 1 \\
\text{IDF} = -\sum_{v \in D} \log df(v)
\]

where the candidate document \(D\) contains the set of visual words \(v_1 \ldots v_n\) and \(df(v)\) is the document frequency of visual word \(v\). We then apply a geometric consistency check by estimating a homography between candidate and query image using RANSAC. We retain only candidates when the number of inliers exceeds a given threshold.

The main difference between our work and others is, that we don’t focus on a ranked list of images as a final result from retrieval. Instead, we want the system to tell us, which object is present in the query image. To that end, we return a ranked list of object clusters instead of images.

### 3.5 Object knowledge from the wisdom of crowds

The database we use in this chapter differs from databases used in the retrieval works in an important point: it is not organized by individual images but by object clusters. We can use this partly redundant information from within the clusters to obtain a better understanding of the objects appearance in an unsupervised manner. Most image retrieval systems in earlier works did neither keep any specific information on the relationships between the items in the

---

2 This corresponds to the set of words document model
database, nor did they exploit it. The most similar work in that respect is maybe [Chum et al. 2007], where initial query results are used to expand a query with additional visual words in order to increase recall. In contrast, in this section, we show how we can use redundancies in the database in order to segment objects from the scene and to create more compact inverted indices. Note that having as compact as possible indices is very desirable when scaling to Internet-scale retrieval and annotation systems with millions of images.

### 3.5.1 Object-specific feature confidence score

The key idea is to use the feature matches from the pair-wise matching step in Section 3.3. Using the information from these matches for each image we can derive a score for each feature, how likely it is to belong to the object that is represented by the images in the cluster. Only features which match to many of their counterparts in other images will receive a high score. In cases where many of the photos are taken from varying viewpoints around the objects, the background will receive less matches. As a result the object is “segmented” from the scene.

We define an object-specific confidence value for feature \( f \) in image \( i \) simply as the number of inlying feature matches stemming from all other images:

\[
\begin{align*}
c_{i,f}^o &= \| \{(u, v) \mid (u, v) \in I_{ij}, j = 1 \ldots N^o \land u = f\} \|
\end{align*}
\]

(3.3)

where \( N^o \) is the number of images in the current object cluster \( o \). \( I_{ij} \) is the set of inlying feature matches for image pair \( i,j \). We can now estimate a bounding box based on a threshold on that confidence value, where the threshold \( t_{i}^o \) for object \( o \) in image \( i \) is defined as

\[
\begin{align*}
t_{i}^o &= \max \left( t_{\text{min}}, \alpha \cdot \frac{\sum_{f=1}^{M_i} c_{i,f}^o}{M_i} \right), \quad M_i = \| \{ f \mid c_{i,f}^o > 0 \} \|
\end{align*}
\]

(3.4)

where \( t_{\text{min}} \) and \( \alpha \) are parameters. We obtained good results with \( t_{\text{min}} = 1 \) and \( \alpha = \frac{1}{3} \). The bounding box is drawn around all features with confidence higher than \( t_{i}^o \), in other words around all features that have a confidence higher than a fraction \( \alpha \) of the mean confidence value.

Examples of the resulting confidence values and estimated bounding boxes are shown in Figure 3.3. Features, which lie within the estimated bounding box
area are more likely to lie on the object and can also be used to create improved inverted indices for later retrieval of the object, as we will demonstrate in the following section.

3.5.2 Better indices through object-specific feature sampling

We can use the gained knowledge on the object clusters to improve the efficiency and precision of our indexing stage.

The estimated bounding boxes can help to compact our inverted index of visual words. Since features which lie outside the bounding box are less likely to belong to the actual object, we simply remove their visual words from the inverted index. This results in a roughly 33% smaller inverted index without any significant loss in precision. (Exact numbers are given in Section 3.7).

An additional improvement in terms of index quality can be achieved by removing irrelevant data. Here, we resort again to the meta data collected alongside the images from Flickr. Besides the associated tags, titles etc. it also tells us which user took the photo. This is valuable information, since it allows us to decide how relevant a given object cluster is for our auto-annotation system. We observed, that clusters which contain images taken only by a single user on a single day sometimes distract the retrieval. This is particularly bad when such
a cluster contains hundreds of near duplicate images, as illustrated by the example in Figure 3.4. We thus simply remove these object clusters from ranked lists returned by the index lookup, or rank them lower.

The last step of the retrieval stage consists of selecting the best object cluster as a final result. The naïve approach of simple voting with retrieved images for their parent clusters has the obvious problem of normalization. Normalizing by cluster size is not feasible either, since some large clusters cover a wide range of viewpoints for a given item. Since only photos taken from a similar viewpoint as the query would match, the normalization by the full cluster would punish these large clusters (which are often the ones of highest quality). A simple but effective solution we found works as follows: only the votes of the $T$ images per cluster with the highest retrieval scores (i.e. TF*IDF) are counted. We obtained good results with $T = 5$.

3.6 **Object-level auto-annotation**

The final annotation stage consists of two steps: bounding box estimation and labelling. The bounding box for the object’s location in the query image is estimated in the same way as the bounding boxes for the database images. To that end, the query image is matched to a number of images in the cluster returned at the top of the retrieval results to refine the initial voting from the retrieval stage. The mean number of votes for all features in the bounding box serves as a score for the bounding box hypothesis.
Since the crawling stage already added location, related text and related content to each object cluster, we can simply copy this information to serve as labels for the query image.

### 3.7 Experiments and Results

We report experiments for all steps of the annotation process. First we evaluate how well our object retrieval stage retrieves the correct object clusters from the database. For the retrieved object clusters we then measure the quality of the object-level annotation.

The experiments were conducted on a large dataset collected from Flickr ([www.flickr.com](http://www.flickr.com)) as described in Section 3.3. We crawled and downloaded over 4 million images at 500px resolution to date. Figure 3.5 shows the hot-spots (i.e. regions with large numbers of photos) encountered by our crawling method. The exact statistics of the set are shown in Table 3.1.

We report results on a subset of of roughly 1 million images, since the processing for the mining stage had not been finished processing at the time of writing this chapter. These images are contained in 63’232 object clusters identified by the mining stage and the estimated bounding boxes cover on average 52% of each image. For this database, we collected a challenging test-set of 674 images from the Internet. We selected a number of object clusters and for each we manually searched Picasa Web-Albums ([www.picasaweb.com](http://www.picasaweb.com)) for images of the same object, ensuring that the images were not contained in our database. By using images from Picasa, we obtain typical holiday snaps shared by users on that platform. The testset is intended to simulate typical photos tourists would take during a holiday trip, including examples taken from varying viewpoints, with partial occlusions of the objects, bad lighting conditions etc. A few examples are shown in Figure 3.6.

<table>
<thead>
<tr>
<th># Images crawled</th>
<th>4’482’582</th>
</tr>
</thead>
<tbody>
<tr>
<td># Images processed to date</td>
<td>996’341</td>
</tr>
<tr>
<td># Object Clusters</td>
<td>63’232</td>
</tr>
</tbody>
</table>

Table 3.1: Dataset statistics.
3.7. Experiments and Results

Figure 3.5: Crawling hotspots.

Figure 3.6: Examples from our test dataset.
3.7.1 Efficiency and Precision of Recognition

To achieve good annotation results, the goal of the retrieval stage must be to return the correct object cluster with the highest possible precision, i.e. we rather retrieve no object at all than a wrong result. The object clusters that were collected during the mining process allow us to reach these high levels of precision, since they contain a large number of images covering the objects from many viewpoints, lighting conditions etc. However, since ultimately we are interested in world-scale auto annotation of holiday snaps, the system needs to be not only precise, but also efficient. Thus, our evaluation emphasizes both criteria.

As introduced in Section 3.4 the retrieval stage is composed of object cluster candidate retrieval followed by geometric verification. In order to evaluate the object cluster retrieval precision, we look at the position of the correct cluster in the retrieved ranked list of candidates. This is illustrated in Figure 3.7. We plot the percentage of query results that contain the correct groundtruth cluster on the y-axis. This value is set in relation to the rank \( k \) in the retrieved list. As a baseline we perform image retrieval on the entire dataset using \( TF*IDF \)-ranking on a 500K visual vocabulary as it is used in other works. The vocabulary is trained on the features of a random subsample of the 1 million images. We compare this to our optimized indices based on the two improvements described in section 3.5. The first optimization – discarding clusters with photos from only one user from retrieval – increases precision by about 5%.

The second optimization reduces the index by about 33% by building the inverted index only from features within the mined object bounding boxes. This significant memory efficiency improvement comes at nearly no loss in precision. We demonstrate the superiority of our approach over random subsampling by also plotting the results achieved with a 66% random subsampling of features. The precision of this method is about 10% lower.

Note that the index can be even further compressed by using standard index compression techniques from text-retrieval, e.g. lossless compression techniques such as Simple-9 [Manning et al. 2008]. Table 3.2 compares the resulting index sizes. It demonstrates how the combination of our bounding-box feature sampling method together with Simple-9 results in a total index size reduction of 67% compared to the naïve implementation where each index entry occupies 4 bytes.
3.7. Experiments and Results

Figure 3.7: Recall in top $k$ results after inverted index, no geometry.

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Index</td>
<td>3.7 GB</td>
<td>1.8 GB</td>
</tr>
<tr>
<td>BB Features</td>
<td>2.3 GB</td>
<td>1.2 GB</td>
</tr>
</tbody>
</table>

Table 3.2: Inverted index sizes in GB. (BB = Bounding Box)

After this step, the ranked list contains between 55% and 80% of the correct clusters within the top 20 and top 1000 positions, respectively. This index lookup typically requires well below 1 second in our current system. The following geometric verification step should move most of these candidates to the top spots.

For this geometric verification step we consider two scenarios. In the first we only use the matched visual words for the homography estimation. Since no more correspondences have to be computed this can be done very efficiently for large sets of images. Our current implementation performs this step for 1000 images in about 2 seconds. In the second scenario we trade speed for accuracy by matching the actual local features of the query to the candidate image features in order to increase the number of correspondences for the homography
3.7.2 Annotation precision

In this section we evaluate how well our system localizes bounding boxes for the retrieved objects. We evaluate the localization precision by measuring the intersection-over-union (IOU) measure for the ground-truth and hypothesis overlap, as it is commonly done also in object class detection. An IOU value greater than 0.5 counts as true positive, any other hypothesis counts as false positive. The evaluation is carried out on all images, that have been correctly recognized from the previous stages of the annotation system, i.e. in this step we measure only how well we can localize the object, given that we already determined its presence in the image. Figure 3.8 shows a ROC plot for this experiment. The curve is generated by varying the confidence (cf. equation 3.4) threshold for the estimated bounding boxes. We reach up to 76.1% localization rate at rather low false positive rate. Note that the false positives are exclusively generated from detections which do not have sufficient overlap with the groundtruth. The geometric verification step after the retrieval stage circumvents the generation of false positives. Figure 3.9 shows final localization and annotation examples. We show the ground truth bounding box in yellow, and our correct detections in green. We also show the Wikipedia article that was considered relevant for the detected object by our system. All objects are also labeled with relevant tag and with a GPS location which are omitted in this figure. Note the wide variety of objects, and the sometimes surprising matches with Wikipedia articles resulting in astonishing annotation estimation. In principle this can be done in near real time (sub 4 seconds) for 1000 images using dedicated hardware like a GPU. On our testset we achieved 0% false positives using a threshold of 8 inliers for the first scenario and 20 inliers for the second scenario. The final recognition rates (i.e. correct clusters at position 1) after the retrieval stage are summarized in Table 3.3. Considering the extremely challenging nature of our testset, these results are of good quality.

<table>
<thead>
<tr>
<th></th>
<th>Feat. Based</th>
<th>Vis. Word Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Index</td>
<td>452 (67.1%)</td>
<td>324 (48.1%)</td>
</tr>
<tr>
<td>BB Features</td>
<td>447 (66.3%)</td>
<td>325 (48.2%)</td>
</tr>
</tbody>
</table>

Table 3.3: Final recognition rates after geometric verification. (BB = Bounding Box)
results. (Remember, that from each Wikipedia article an image matched to our object cluster, which we used to verify the content assignment as described in section 3.3.)
3.8 Conclusions

In this chapter we presented a full auto annotation pipeline for holiday snaps. The complete system functions fully automatically starting with data crawling up to object-level annotation with bounding box, relevant tags and Wikipedia articles, as well as GPS location. The system design allows scaling to millions of images in the reference database. To achieve this functionality, we combined a multi-modal data mining method for community photo collections with state of the art object retrieval based on visual vocabularies. The mining method groups photos of objects into object clusters which serve as exemplar models for items in the database. These object clusters allow us not only to recognize query images for annotation from a wide variety of viewpoints, but also to automatically localize the object in database images. We showed how to exploit this information to reduce the index size without loss in retrieval precision,
enabling further scaling. The system was evaluated on challenging test-data and a large database in terms of correct recognition and annotation of objects as well as their localization within the query image.
4 Improving Large Scale Autoannotation

4.1 Introduction

Recognition, reconstruction and analysis of 3D scenes are topics with broad coverage in the Computer Vision literature. However, in recent years the enormous amount of photos shared on the Internet has added a few new twists to these research problems. On the one hand there is the obvious challenge of scale, on the other hand there is the benefit that photos shared online usually come with meta-data in form of (geo-) tags, collateral text, user-information, etc. Besides the interesting research that can be done with this data, they also open doors for real-world deployments of computer vision algorithms for consumer applications, as recent examples from 3D scene browsing [Snavely et al. 2006], or face recognition [Stone et al. 2008] have shown.

Consequently, a number of works have started to exploit these cross-media data in several ways [Zheng et al. 2009, Li et al. 2009, Agarwal et al. 2009, Crandall et al. 2009, Hays and Efros 2008, Kalogerakis et al. 2009, Quack et al. 2008, Schroff et al. 2007, Simon et al. 2007, Snavely et al. 2006, Torralba et al. 08]. [Quack et al. 2008] have used a combination of GPS tags, textual and visual features to identify labeled objects and events in data from community photo collections such as Flickr. [Crandall et al. 2009] have done similar experiments, but at even larger scale (up to 10s of millions of photos) and analyzing temporal movement patterns of photographers in addition to GPS, textual and visual features. In the previous chapter we have also exploited

1www.flickr.com
4. Improving Large Scale Autoannotation

these cross-media data collections from the Web in order to build applications for auto-annotation.

Also in the 3D reconstruction field there has been a long-lasting interest in reconstruction the whole world in 3D, and not astonishingly, community photo collections nowadays serve as a data source for this purpose as well [Agarwal et al. 2009, Simon et al. 2007, Snavely et al. 2006]. In spite of the different target applications, all these works have one theme in common: the underlying data structures are clusters of photos depicting the same object or scene, accompanied by some cross-modal data, such as (geo-)tags etc. In this work we are particularly interested in clusters of consumer photos showing “places”. Places include any geographic location, which is of interest to people, such as landmark buildings, museums, mountain peaks, etc. Similar to most works cited above, in a first step we also cluster images in order to identify relevant places. While attention has recently been directed towards harvesting larger and larger collections of data, in this chapter we want to take a step back and look at the collected image clusters in more detail. The objective is to investigate if and how basic knowledge about the 3D scene in combination with analysis of cross-media data is helpful towards improving the quality of the database of places as well as the performance of applications building on top of the database. More precisely, we show how

- cross-media retrieval helps identifying missing information for small clusters.
- scene analysis helps removing redundant data in large clusters.
- those measures affect performance of object recognition and 3D reconstruction applications relying on the database of image clusters.

In other words, if we take the analogy of a web search engine for hypertext documents, we focus on the crawling and indexing part of the system. While in the hypertext retrieval community this topic is well documented, in the Computer Vision field most work has focused on the retrieval side of things [Chum et al. 2007, Chum and J.Matas 2008, Jegou et al. 2008]. For instance, Chum et al. [Chum et al. 2007] could show how to improve retrieval precision using query expansion, using an algorithm which operates mainly at retrieval time.

With our improvements on the crawling and indexing stages of the pipeline, we can indirectly achieve significant improvements in an object recognition
setting. We focus on the object recognition task, since there are clearly defined evaluation metrics available. In addition our contributions are valuable for unsupervised 3D reconstruction as well, however, the improvement in this application is in general less easily quantified, but easily visualized. Most importantly, for both scenarios, every proposed improvement happens offline and all the processes we show in this chapter are fully automated.

The chapter is structured as follows: Section 4.2 describes our basic methods for image cluster mining and object recognition. The core of our methods for automated cross-media cluster analysis and optimization follows in Section 4.3. Experiments and analysis of the effects of optimization on retrieval tasks follow in Section 4.4. Section 4.5 concludes the chapter.

### 4.2 Mining and Recognition of Objects

As discussed in the introduction, harvesting photos from online services for landmark mining, recognition or 3D reconstruction has been addressed in a number of recent works. We build on some of those ideas in order to construct our own image mining pipeline. We also introduce the object recognition methods, which we apply on top of the mined data.

#### 4.2.1 Object Mining

Several ways have been proposed to collect data from online photo collections in order to solve computer vision tasks. They either start out by querying with certain keywords such as "Rome", "Venice" [Simon et al. 2007, Snavely et al. 2006, Li et al. 2008, Philbin and Zisserman 2008], or with collecting geotagged photos [Crandall et al. 2009, Quack et al. 2008]. For bootstrapping our system we chose the latter strategy.

Unlike in the previous chapter, in order to harvest photos from Flickr based on their geo-tags, we overlay several geographic quad-trees over the world and retrieve the number of photos in each tile. Each of the trees is initialized by a country’s geographic bounding box coordinates. Recursively this initial area is then subdivided as follows. We retrieve the number of photos in the current area from the Flickr API. When the number of photos is higher than a threshold (250 in our implementation), we split the area into 4 tiles of equal
size and repeat the process for each tile. The recursion stops when the threshold for the number of photos is reached. In addition, the dimension of the tile in meters also serves as a second stopping criterion: the process returns when the tile’s extent is less than $200m$ (on the smaller side). The outcome of this is shown in Fig. 4.1. Photos are then downloaded for all child leaves, and the photo clustering is also distributed based on the child leaves of the geographic quadtree. For clustering photos, we then proceed as proposed in [Quack et al. 2008] in three steps:

1. Match photos pair-wise using local image features (we use SURF [Bay et al. 2006]).

2. Build a set of image similarity matrices. We create one matrix per geographic leaf tile. The similarity is the number of inlying matches after RANSAC filtering of feature matches for each similar image pair.
3. Cluster the photos using single-link hierarchical agglomerative clustering.

For each cluster we keep its photos including their meta data (tags, titles, user information etc.) for further processing. Very similar to [Quack et al. 2008] we observed that the image clusters usually represent one common object, but covered with photos from various viewpoints and under various lighting conditions etc. Thus, we think of each cluster representing one particular object and consider the images of a cluster to form an exemplar based object model.

Qualitatively, we think our crawling method ends-up with very similar data like [Quack et al. 2008], but is significantly more efficient ([Quack et al. 2008] scans the world in evenly distributed tiles of equal size, in effect querying a lot of empty cells unnecessarily.) We believe our crawling approach is also beneficial over [Crandall et al. 2009], since we can split the clustering problem into smaller parts, and the tree based approach is directly “pulled” towards densely populated areas already while collecting the data. In contrast, [Crandall et al. 2009] is one huge clustering problem. Finally we crawled a significantly larger dataset than in the previous chapter with our quadtree method (17 million images w.r.t. 4 million), to be able to compare our results in terms of object recognition with theirs as a baseline, for the remainder of this chapter we conduct all our analysis on the same data (the dataset is available from the authors web-site).

4.2.2 Object Recognition

Like in the previous chapter, given a query image depicting a landmark, the goal is to identify and label this object based on the information aggregated in our reference database of image clusters. (In contrast to image/object retrieval [Nistér and Stewénius 2006, Philbin et al. 2007, Sivic and Zisserman 2003], where the expected outcome is a ranked list of similar images or images showing the same object as the query, sorted by similarity).

Like in the previous chapter, at the lowest level, our object recognition system builds on “standard” visual word based image retrieval. Local image features [Bay et al. 2006] are clustered into a visual vocabulary of 1 million visual words using approximate $k$-Means (AKM) [Philbin et al. 2007]. An initial top-$n$ list of the $n$ most similar images in the database in terms of set intersection is
efficiently computed using an inverted file structure. We then use RANSAC to estimate a homography between the query image and every image in the top-$n$ list. Candidate images for which the RANSAC estimate yields less inliers than a threshold (13 in our implementation) are discarded. We then simply let the image with the highest number of inliers to identify the object in the query image. This is in contrast to the previous chapter, where the images in the filtered top-$n$ list are used to vote for “their” object.

4.3 Cross-Media Cluster Analysis

The main object of study for the remainder of this chapter are the image clusters mined from the Internet as described in the preceding section. Given our target applications — object recognition or 3D reconstruction — we can now analyze and improve the image clusters in several aspects. The first and most obvious aspect is cluster size. Intuitively, objects which are represented by a smaller cluster should be more difficult to recognize, since they may lack images taken from an important viewpoint. Fig. 4.2a shows a (histogram) plot of the cluster size versus recognition rate. It confirms that recognition tends to be more successful for larger image clusters. (Detailed results for recognition are given in Section 4.4 of this chapter.) Further, as illustrated in Figure 4.2b, it seems that the cluster size distribution follows a power law: $p(\text{Cluster Size}) \propto \frac{1}{\text{Cluster Size}^\alpha}$ with a maximum likelihood estimate of $\alpha_{MLE} = 1.41$. Such distributions are extremely heavy tailed, and thus imply several characteristics. For instance, one should note that it is unreasonable to consider an average cluster size, since the expectation value diverges for $\alpha \leq 2$. Further, from the power law distribution also follows that the majority of clusters is small, but due to the heavy tail quite a few clusters are disproportionally large. It stands to reason that these extremely large clusters carry a large amount of redundant information. Thus, in the following, we investigate the effect of expanding small clusters with additional (non geo-tagged) images, and propose strategies for reducing redundant information contained in very large image clusters.

4.3.1 Expansion of small clusters

Even though an increasing number of digital images shared online contain geo-tags, owning a GPS-equipped camera is still not standard today. Consequently,
4.3. Cross-Media Cluster Analysis

(a) Objects represented by small clusters yield lower recognition rates.

(b) Many clusters are small

Figure 4.2

a significant fraction of clusters mined using an approach relying on geo-tags, consists only of a handful of images (Fig. 4.2b). In fact, in our dataset 81% of all clusters contain 10 images or less. For some places this is simply because they are not popular enough. Note that with keyword based mining we would not have been able to find such rare objects in the first place — a list of terms so extensive that it covers such locations is simply not available. But even for much-visited locations many images can lack GPS tags, if the location is e.g. inside a building. In order to enrich such small clusters, we propose to use a cross-media crawling method. First, text queries are generated using the tags associated with an existing image cluster. To that end, we follow the approach taken by [Quack et al. 2008], where text queries are automatically created from the meta-data of the photos in each cluster. They then use these queries for crawling Wikipedia articles intended to serve as descriptions for image clusters. In order to generate the text queries automatically, the authors propose to use itemset mining [Agrawal et al. 1993] to form frequent combinations of tags for each cluster. We follow the same approach, but query the WWW for images instead for Wikipedia articles. For the remainder of this chapter we call these automatically generated text queries itemset queries. The itemset queries are used to query common Google for additional photos. The retrieved images are then matched against the images inside the cluster, again by estimating a Homography using RANSAC and SURF [Bay et al. 2006] features. Matching images are added to the cluster. Match vs. no match is determined based on an inlier threshold of 15 feature correspondences. This procedure is illustrated in Fig. 4.3.
4. Improving Large Scale Autoannotation

Figure 4.3: Cross-media expansion of image clusters: 1) starting out from clusters of images (clustered by their visual similarity with the help of geo-tags for efficiency), we use itemset mining to generate text queries from frequent tags. 2) in order to retrieve additional images thus expanding the image cluster with additional information, 3) and finish with a verifying matching based on visual similarity. We also show the Cluster Match Rate (CMR) for each itemset query (see Section 4.3.2.)

4.3.2 Efficient Itemset Query Selection

It turns out, that for a surprisingly large amount of clusters additional images can be retrieved (96% of clusters in our test dataset have been expanded by at least one image). Furthermore, one should note that as shown in Fig. 4.2a this procedure is more likely to be successful for larger clusters than for smaller ones. Obviously, when applying such an automatic query generation approach for a large amount of data with many clusters, the number of text queries can reach a level, where efficiency considerations become crucial. (Each cluster can generate dozens or even up to hundreds of different itemset queries). Unlike other resources like bandwidth or storage, the amount of HTTP requests that can be made to a public service like Google is often limited. Furthermore, results from search engines are returned aggregated to pages. Each page usually contains only about 20 images and requires one additional HTTP request to retrieve it. While the prices of resources like computation power (Moore’s Law), bandwidth (Nielsen’s Law) and storage (Kryder’s Law) drop exponentially over time, this most likely does not imply the same exponential increase in the number of queries that can be made to search engines. (They are already confronted with a rapidly growing user base.) So, unless one has the resources
to crawl the entire Internet in order to avoid public search engines, it is of great interest to minimize the number of queries required. However, if an itemset query is not very specific (e.g. “town hall”, compare Fig. 4.3), it might lead to the retrieval of a large number of images, which do not have anything in common with the object in the cluster, and consequently won’t match to its images. In other words, to be efficient, we have to find a way to automatically select itemset queries which have a higher probability of returning relevant images.

As a basic measure for how successful an itemset query is in retrieving additional images of the object, we define first the cluster matching rate (CMR).

\[
CMR = \frac{\text{# Matching images}}{\text{# Retrieved images}}
\]

This is a straightforward choice, which records for a given itemset query the fraction of retrieved images that match to the images in the database cluster. While CMR is useful to determine the quality of an itemset query once all images have already been retrieved and matched, an efficient approach should discard itemset queries with low CMR well before that. This could entail estimating the CMR, which in turn would require in the order of \((1/CMR - 1)\) images. Thus, the lower the CMR of an itemset query, the more images we would have to download before we can reject it. By comparing the improvement in recognition quality on the test set when considering all queries vs. the improvement when only accepting queries with a CMR above a given threshold, we find that the largest improvement comes from queries with a CMR between 0.01 and 0.1. This is shown in Fig. 4.8. In other words, we might have to download at least 100 images before we can safely reject any itemset query. We can, however, exploit an observation made by [Quack et al. 2008, Gool et al. 2009]. The authors used text queries in order to retrieve Wikipedia articles intended as descriptions for the image clusters. The trick they came up with, is to verify the retrieval result by matching images from the articles to the source cluster. They found that itemset queries yielding articles containing images matching the cluster have a higher probability of yielding matching images from other sources as well. This could be a crude indicator to a-priori assess an itemset query’s CMR. In order to test this hypothesis, we downloaded and indexed a dump of all English Wikipedia articles and their images. Then, as illustrated in Fig. 4.4, for any given cluster, we query both the text index with the itemset queries and the image index with the cluster’s images. The text index returns ids of Wikipedia articles with words matching the itemset query, while the image index returns ids of Wikipedia articles with images matching
4. Improving Large Scale Autoannotation

Figure 4.4: A priori estimation of CMR using a local copy of Wikipedia. An inverted text index and an image index are queried simultaneously. The result sets are intersected in order to determine if the text could have yielded any useful images.

4.3.3 Reduction of large clusters

While small objects that are only modeled by few images in their respective image clusters are more difficult to recognize, having too much data is not a blessing either. Unusually large amounts of photos are often collected at popular tourist destinations such as Notre Dame de Paris, or the Eiffel Tower. Many of these photos contain redundant information, increasing in an image retrieval scenario, unnecessarily the size of the inverted index. Furthermore, since our method from Section 4.3.1 allows for augmenting almost any cluster by an arbitrary amount of images, we desire to find a method that purges the redundant information, while leaving complementary information untouched.
Figure 4.5: Left column: example of a free-standing 3D object which can be photographed from many viewpoints (top) vs. one which is visible nearly from only a single viewpoint (and even only 2-dimensional in this particular case). Right column: example of cluster reduction. The full single-linked matching graph is shown on the top left. A complete-linked section which is removed on the top right. Bottom left: images from the removed complete-link segment. Bottom right: images which stay in the cluster.

Note that it is a-priori also unclear what “a good” number of images would be for an arbitrary cluster, since it strongly depends on the 3D scene structure of the given object. This is illustrated in Fig. 4.5. The object on the top left is a free standing structure which can be photographed from arbitrary viewpoints, so an image cluster which serves as a model for this object has to contain many images. In contrast, the example on the bottom left is the extreme case of a painting in a museum, which can be seen from a small number of viewpoints only, so fewer reference images are necessary to “describe” the object. In fact, while 3D scene structure makes it impossible to generalize to a “good” cluster size, it is at the same time key to attack the problem of extraordinary cluster size. It turns out, that with some simple 3D scene analysis we can compact the clusters in both an effective and scalable manner. Remember, that the image clusters were created using single-link clustering (Section 4.2). We now decompose these single-link clusters into several overlapping complete-
link clusters. Note the definition of single-link and complete-link criteria in hierarchical agglomerative clustering [Webb 2002]

\[
single-link: \quad d_{AB} = \min_{i \in A, j \in B} \, d_{ij} \quad \text{complete-link:} \quad d_{AB} = \max_{i \in A, j \in B} \, d_{ij}
\]

where for clusters \( A \) and \( B \) the indices \( i, j \) run over the images in the clusters and \( d_{ij} \) is the image distance measure that is proportional to the inverse of the number of inliers. Complete-link requires that all image pairs in a segment are fully connected to each other. In our setting this is the case if all image pairs match to each other, which means that they are all taken from a very similar viewpoint. This procedure is illustrated in Fig. 4.5. Then, for every complete-link cluster with more than 3 nodes, we find the node with the minimum edge-weight-sum (i.e. the image most similar to all its neighbors) and remove all other nodes. In essence it is an idea similar to the scene graph in [Snavely. et al. 2008], but can here be derived with standard tools using the already calculated distance matrix.

When we remove these highly similar images from the index we automatically remove highly redundant information, while guaranteeing that we keep relevant data. As demonstrated in the experiments in section 4.4.1 this procedure reduces the index size for retrieval tasks significantly, without affecting precision.

### 4.4 Experiments and Results

For all our experiments we used the same dataset as in the previous chapter. The dataset consists of roughly 1 Million images from Flickr that were clustered into 63'232 objects and a test set of 676 images which are associated with 170 of the 63'232 objects. The goal is to correctly identify which object in the database is shown in the images of the test set. The percentage of images correctly associated with their object serves as an evaluation metric. We first report evaluation results on overall recognition performance including the overall effects of cluster expansion and cluster reduction. Finally, we demonstrate that our additionally mined images can be vital in 3D reconstruction.
4.4.1 Object Recognition

We compare our object recognition system to the approach of the previous chapter. On the benchmark dataset we achieve similar baseline performance, as shown in Fig. 4.6. Adhering to the original evaluation protocol we consider the percentage of test images for which the correct object is returned in its top-$n$ candidate list vs. the toplist size $n$. This is an upper limit for the recognition rate after geometric verification. We then applied our cluster expansion and reduction methods to the image clusters in the benchmark dataset. For each of the 170 clusters in the testset we generated itemset queries in order to retrieve additional images for cluster expansion according to the methods described in Section 4.3.1. We carried out experiments with 3 major image search engines and found that using Google yielded the best results. For every itemset query we retrieve the first 420 images returned by Google to expand our object models. Fig. 4.6 clearly shows that expanding clusters substantially improves recognition. However, since we only expand clusters that are relevant to the test dataset, we created an unfair situation: the expanded clusters now have a higher probability of randomly occurring in a top-$n$ list. We thus plot the chance level in Fig. 4.6 (dashed lines) for each expanded index. The comparison highlights that the observed improvement is not simply an artifact of an increased chance level.

A summary of the achieved improvements over the baseline is given in Table 4.1. The first two columns show cluster retrieval results with bag of visual words lookup for finding the correct cluster in the top $n$ ranked results. The third column shows results for identifying the correct object/cluster on the first rank, using geometric verification. To that end the top ranked 1000 results after bag of words lookup were verified by estimating a Homography mapping between query and retrieved images using RANSAC. For this last task, we achieve 14.8% improvement over the previous chapter, when using our cluster expansion method.

We also applied the reduction strategy from Section 4.3.3 to the baseline index, as well as to the expanded index. In both cases we find that our strategy for “purging” unnecessary images does not significantly influence recognition quality as demonstrated in Fig. 4.6. However, it reduces the inverted index file size significantly, as shown in Table 4.2. One can also observe that the relative reduction in size is much larger for the expanded index. This is due to the fact, that retrieving additional images via itemset queries more often leads to
Table 4.1: Absolute number of testsets with a correct cluster within the top-100 and top-1000 list. The last column is the absolute number if test images correctly labeled after geometric verification of the top-1000 list.

<table>
<thead>
<tr>
<th>Description</th>
<th>top-100</th>
<th>top-1000</th>
<th>top-1 Geo.Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>63.4%</td>
<td>78.6%</td>
<td>73.52%</td>
</tr>
<tr>
<td>Expanded</td>
<td>73.0%</td>
<td>86.1%</td>
<td>78.1%</td>
</tr>
</tbody>
</table>

Table 4.2: Index size comparison for indices built from the original clusters vs. reduced clusters.

<table>
<thead>
<tr>
<th>Description</th>
<th>Original</th>
<th>Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1.5GB</td>
<td>1.3GB (−13%)</td>
</tr>
<tr>
<td>Expanded</td>
<td>2.1GB</td>
<td>1.5GB (−29%)</td>
</tr>
</tbody>
</table>

duplicates or near duplicate images compared to images retrieved using GPS queries during the initial crawling of clusters.

4.4.2 Cluster reduction statistics

It is of interest to take a closer look at which clusters are reduced using our method, and which clusters contribute most to the overall reduction in database size. Intuitively larger clusters should more often be affected by a reduction for which Figure 4.7a gives clear evidence.

However since most clusters are rather small, even though the chance of a small cluster being reduced is lower, they contribute the most to overall database reduction. This is shown in Figure 4.7b, where the number of images removed from the index for all clusters smaller than a threshold \(N\). One can clearly observe that the majority of the overall savings is due to the small clusters.

4.4.3 Efficient Itemset Query Selection

In total 2030 itemset queries were generated for the testset of 170 image clusters. As mentioned in Section 4.3.2, on the fly estimation of CMR based on retrieved images is not beneficial, since at least 100 images have to be retrieved...
4.4. EXPERIMENTS AND RESULTS

Figure 4.6: Expanding clusters with additional images from Google significantly improves top-$n$ score. Reduction of clusters (with “purging” of complete-link segments) does not affect performance, neither for the baseline nor for previously expanded clusters. Improvements can not be attributed to chance, as the dashed lines show.

before an itemset query can be safely discarded. However we can use the estimated CMR (cf. Section 4.3.2) as an indicator if an itemset query is useful or not. This is demonstrated in Fig. 4.8. We found that if we do not discard itemset queries with an estimated CMR above 0.01% we retain about 75% of the original improvement. For the test dataset only 40% of all queries fulfill this requirement, however as visible in Fig. 4.8 these queries alone are responsible for the 75% improvement in recognition quality.
Figure 4.7: (a): Probability of at least one image being removed from a cluster vs cluster size. (b): Cumulative sum of images removed, considering all clusters of size zero to N

4.4.4 3D Reconstruction

So far we focused on demonstrating the outcome of the proposed cluster expansion and reduction methods on an object recognition task. As mentioned earlier, this is due to the easy quantification of the evaluation. However, the same methods can also be beneficial in a 3D reconstruction scenario. The outcome of image based 3D reconstruction is highly dependent on the images used as input. In essence, a large number of high-resolution images taken from a wide variety of viewpoints is desired.

That a simple keyword search or geographic query yields enough images for a decent reconstruction of an arbitrary object is far from a given. Such a strategy in fact only works for a fraction of all landmarks. Even for popular sites, manual keyword search is not trivial, because it is not feasible to efficiently come up with so many appropriate keywords. For less famous landmarks, the situation is even more dire. Even a keyword search with the precise description of the object may not yield enough useful images nor would GPS-based retrieval.

In such cases every single image matters, and a couple of additional images of high quality may dramatically change the outcome of the reconstruction. In this section, we briefly demonstrate with two examples that our cluster expansion method yields additional images crucial for 3D reconstruction. Using
Figure 4.8: Considering top-100 accuracy, we compare the overall improvement to the baseline obtained when considering only queries with a CMR above a certain threshold (blue line). The red line shows what happens if we discard itemset queries if and only if their CMR is below a certain threshold and their estimated CMR is below 0.01%.

the publicly available ARC3D [Vergauwen and Van Gool 2006] reconstruction tool, we compare the 3D reconstruction of the originally mined image clusters of the previous chapter to the reconstruction based on our expanded clusters. From a set of uncalibrated images, ARC3D generates dense, textured depth maps for each image. Input images are uploaded and processing is performed remotely on a cluster, so that results can be obtained within short time. As demonstrated in Fig. 4.9, additionally mined images clearly help in reconstructing more complete 3D models.
4.5 Conclusion

We have shown a fully automated cross-media method to improve the quality of reference databases for object recognition. Small image clusters were enriched with additional information by automatically generating text-queries from image meta-data. Redundant information was purged from large clusters by a simple graph based approach. The combination results in better performance and higher efficiency (in index size) for object recognition tasks on recent benchmark data for object instance recognition. We have also shown that it is possible to exploit the wisdom of crowds to a-priori determine if a potential text query may be useful for retrieving additional images. Finally, while this chapter focused on object recognition, the cluster expansion method would be also valuable for unsupervised 3D reconstruction.
Mobile Visual Search

5.1 Introduction

In recent years, research in Augmented Reality (AR) has made significant progress towards real world consumer applications. One of the main drivers for this development have been the increased processing capabilities and multimedia features of mobile phones. The most advanced class of devices (so called smartphones) nowadays come equipped with high-resolution touch screens, cameras, accelerometers, GPS, compass, etc. Such devices make ideal platforms for consumer-oriented AR applications (video see-through), since they are widely available, cheaper and also more discrete than, say, wearing head mounted displays (HMD, optical see-through).

Most of the AR applications for phones have focused on the visual modality, overlaying virtual objects on the real world seen through the live camera feed on the mobile phone’s screen. The basic concept consists of identifying real world objects on the screen, tracking them, and then augmenting the scene with artificial objects. Tracking is often combined with some estimation of the correct 2D or 3D world coordinates for proper placement of augmentations in the scene. Applications can then be further discriminated into those, which build on artificial markers in order to identify objects [Rohs and Gfeller 2004, D. Wagner and Schmalstieg 2003, Wagner et al. 2008a], or those, which use “natural” image features [Ferrari et al. 2001, Klein and Murray 2008, Klein and Murray 2009, Skrypnyk and Lowe 2004]. Both kinds of application have been demonstrated to work in real-time on PC’s for quite a while [Ferrari et al. 2001, Skrypnyk and Lowe 2004], but have gone through a renaissance with their adaption to mobile computing platforms in recent years [Klein and Murray 2008, Klein and Murray 2009, Adams et al. 2008]. Here, especially
the discovery of more robust and efficient local features [Bay et al. 2006, Lowe 2004, Rosten and Drummond 2006] has fueled the advance of markerless approaches.

In a parallel development, the computer vision community has made astonishing progress in visual object recognition, both in terms of precision and scalability [Chum and J.Matas 2008, Jegou et al. 2008, Nistér and Stewénius 2006, Philbin et al. 2007, Sivic and Zisserman 2003]. State of the art methods allow for retrieving specific objects (such as products, buildings, etc.) from databases with millions of items in a matter of seconds. In fact, much of this advance can also be attributed to the same improved local features, which are responsible for the improved tracking performance on the client.

Combining those two strands of work has the potential for very exciting AR applications. To that end, in this chapter we propose a hybrid approach, which
delegates the object recognition task to a server, and carries out tracking on the client-side, \textit{i.e.} on the mobile phone. This has several advantages. Firstly, it allows to retrieve objects from very large databases in near real-time, as opposed to keeping the database on the client. Second, it still allows for interactive usage on the client, as opposed to sending only single images with the click of a button. Third, while being interactive, the approach also limits the amount of communication with the server, as opposed to the extreme case of transmitting live video to the server. This is particularly important considering today’s mobile data transmission costs, especially when it comes to roaming charges due to the usage of data intensive applications abroad.

Furthermore, in comparison to AR approaches that strongly rely on GPS and sensor input like [lay, Kähäri and Murphy 2006, wik, Takacs et al. 2008], basing a system on visual recognition has the advantage, that both stationary objects (such as landmark buildings) and non-stationary objects (such as products, billboards, print media \textit{etc.}) can be augmented seamlessly. The advantage over marker-based approaches is also quite obvious, as no markers have to be placed, which allows widespread application. On the other hand it is evident, that the usefulness of such a system is constrained by the size of the object database on the server side. To cope with this challenge, we build on the system presented in the previous chapters for landmarks and a commercial API for product recognition [koo].

With all these components in place, the research we present in this chapter relates to several recent lines of work in both commercial and scientific context. Overall, taking into account the server-side modules our system is built of, Google Goggles [goo] is probably the product that is most similar to our system. However, to this date, the client-side of Google Goggles does not allow “live” tracking and augmenting of objects, \textit{i.e.} recognition is triggered with a manual shutter release. Adding tracking adds the important option to incorporate gesture like Human-Computer Interaction, \textit{e.g.} to indicate regions of particular interest. Also, in terms of server-side recognition, besides Goggles our system must be among the currently largest deployments, using roughly 12 million images for landmark recognition (following the approach of [Gammeter et al. 2009]) and about 10 million media covers using the public API of kooaba [koo].

In terms of client-side tracking this chapter relates to several recent advances reported in that field [Klein and Murray 2008, Klein and Murray 2009, Ta et al. 2009, Wagner et al. 2008b]. As we will show, our tracking is somewhat
simpler than many of the recent methods (e.g. it does not allow for 3D scene estimation). However, this comes with a benefit of efficiency, and furthermore, we argue that for many AR applications 2D augmentation is fully sufficient. A further contribution on the client-side tracking element is the integration with sensors. Opposed to other works, we do not attempt to fuse the information from visual and sensor modalities, but use the sensors to reset the visual tracking when needed.

In summary, the main contributions of this work are:

- An efficient combination of client-side object tracking and server-side object recognition.
- The complete integration of a server-side object database covering millions of objects.
- Memory efficient geometric verification method for state of the art large-scale object retrieval methods to fulfill near real-time requirements for AR.
- The integration of sensor data in order to reset client-side tracking.
- A fully functional implementation of the system on the Android platform.

With these contributions our system makes a significant step ahead from earlier prototype applications, bringing vision-based AR for mobile phones significantly closer to real world applications.

The remainder of this chapter is organized as follows: Section 5.2 gives an overview over the system architecture. Sections 5.3 and 5.4 describe the server-side recognition and client-side tracking, respectively. Implementation details are given in Section 5.5 and experimental results are shown in Section 5.6.

### 5.2 System overview

As motivated in the previous section, we propose a system, which delegates recognition tasks to a server and executes tracking operations on the client. The purpose of this is to provide large scale object recognition coupled with a responsive user interface for mobile AR. When a relevant object is seen through the mobile phone’s screen it should immediately be overlaid with relevant information, ideally without noticing the processing of a search request in the
5.2. System overview

Figure 5.2: The mobile phone sends recognition requests to an external service. Once information on a recognized object is returned, the phone tracks the position of said object using a combination of vision and sensor based trackers. Whenever a tracked object is “lost” a new recognition request is issued to reinitialize tracking.

backend. To that end we actively try to minimize the number of requests sent to the server by determining if a new object may have entered the camera’s field of view. Figure 5.2 gives an overview over our system. The tracker running on the client initiates a request to a recognition service, transmitting both a frame grabbed from the camera feed and optionally GPS coordinates over a HTTP connection. The moment for sending a request is chosen based on a heuristic in the tracking algorithm, which will be explained in detail in Section 5.4. In fact, in our application the client sends the request to two recognition services in parallel, namely one for stationary objects (landmarks) and one for non-stationary objects (products, media covers). Details for this step will be given in Section 5.3. The server’s response consists of XML data containing the information about the recognized object (title,id) and the location of the detection in the query frame (bounding box coordinates). The bounding box coordinates are then used by the client to (re-)initialize the tracker, and the title is used to label the augmentation on the screen. The id of the object can be
used to obtain additional information about it on user’s request. This operation initiates a request to a dedicated object information web-service, which will not be explained in further detail in this chapter. The following sections will describe the individual components of the system in more detail.

5.3 Server-side object recognition

The tasks of the server side recognition module are to accept query images, to identify any objects present including their location in the image, and return the response to the client. To that end we build on state of the art approaches using visual vocabularies of local image features [Gammeter et al. 2009, Nistér and Stewénius 2006, Philbin et al. 2007]. More precisely, for landmark recognition, we follow very closely the system described in the previous chapters, which in turn merges a data crawling method with the scalable object recognition method of [Philbin et al. 2007]. We then combine our own service for landmark recognition with the media recognition service offered by the company kooaba. Their API gives access to recognition for a couple of million media covers (books, CDs, DVDs) and is available online [koo]. We assume that kooaba’s system also builds on local features and visual vocabularies, thus we will focus on summarizing the landmark recognition in this section.

5.3.1 Landmark recognition

A visual recognition service for AR applications, which is able to identify objects such as landmark buildings requires both a database of images, covering each object from several viewpoints, and a scalable and near real-time retrieval method on top of it. In addition, for each object in the database some description such as titles, related web-links etc. should be available.

We have proposed such a system in the previous chapters, in the context of auto-annotation for holiday photos. Ultimately, this application is similar to AR except that the query happens from a mobile device and demands for a close to real-time response. Thus, for this chapter, we reuse the system described in the previous chapters and then focused on improving its response times. We briefly summarize the previous chapters and then explain our improvements.
Crawling and clustering of data. In order to collect a sufficient amount of images for a large number of landmarks around the world, we proposed to crawl geotagged images from Flickr. All photos, which are geographically close to each other are then also checked for their visual similarity (based on matching with SURF features [Bay et al. 2006]) and then clustered.

The outcome of this procedure is a set of clusters, where most of them represent some landmark. An example cluster is shown in Figure 5.3. Once clusters have been formed based on visual similarity, a post-processing step is executed. Among other things, it determines labels and related content for each object by resorting to the meta data of its cluster. Also, for each image of an object cluster, the bounding box of the object’s position is calculated. For a more detailed description of these steps the interested reader is referred to the previous chapters. Following this approach, we collected around 12 million images, which we clustered into roughly 300’000 objects.

Cluster retrieval. For all images in the cluster database their SURF features are quantized using a visual vocabulary of 1 million visual words which was learned using approximate k-means [Philbin et al. 2007]. Every image in the database is then represented as a set of visual words. For every incoming query image the same procedure is applied. The query-set of visual words is matched against all database visual word sets using set intersection as distance measure.
Efficiency for this step is obtained by using an inverted file structure. For the closest 500 candidate images a geometric consistency check is performed. This is done by means of a RANSAC estimation of the homography mapping of feature matches between query and database candidate. The final matching score for each of the 500 candidate images is derived from the number of inliers.

Here, in contrast to many other implementations, we do not take the matching visual words as correspondence pairs for the homography estimation. Instead we reestimate the correspondences based on the second nearest neighbor distance ratio of the original features as proposed in [Lowe 2004]. This approach yields far better retrieval accuracy. A drawback is that for each query 500 SURF feature files need to be loaded into memory from disk.

The final step is a voting procedure, where the retrieved database images vote for their clusters and the cluster which receives the most votes is selected as the final result.

**Optimized response times with compressed features.** The geometric verification step turned out to be the bottleneck for achieving AR compatible response times. Assuming a single file access takes roughly 10 ms, then geometric verification alone would take at least 5 seconds. So, instead of loading features from disk, we compress them using a product quantizer [Jégou et al. 2011a] in order to be able to keep the features in memory.

The main idea of [Jégou et al. 2011a] is to decompose the feature space into a Cartesian product of low dimensional subspaces and to quantize each subspace separately. A feature vector is then represented by a short code composed of its subspace quantization indices. In our implementation first the entire 64-dimensional feature space is quantized using only 8 centroids (3 bits). Then, for every feature vector in the training set we subtract it’s closest centroid and the marginalized features are used to train a product quantizer. To that end, we divide the 64-dimensional feature space into 16 4-dimensional subspaces and quantize features in each subspace using k-means with 1024 centroids (i.e. 16 × 10 bits). Thus, a feature vector is now represented by a compact 160 + 3 bit code instead of 64 float or integer values.

For one million images we require now approximately 30 GB of storage instead of 300 GB. This training procedure was carried out on the features of 5000 images. During feature correspondence search in the geometric verification
step, we only consider features as matching candidates if their 3 bit coarse quantization indices match. We also found that in contrast to image retrieval with large vocabularies it doesn’t matter whether the quantizer is learned on the database images themselves or on an independent set.

With this adaptation, we gain a significant speed improvement over the previous chapter. In the original implementation, geometric verification is performed using uncompressed feature files that are (due to their size) loaded from disk. The resulting recognition times would be on the order of 10 seconds, which is not sufficient for an AR application. Using our in-memory quantized features we achieve recognition times strictly under 2 seconds.

5.4 Client-side object tracking

Once an object has been recognized by the server and the response has been received by the mobile phone, a virtual label is attached to the object. The label’s on-screen position has to be updated for every frame, i.e. we need to track the object. Thanks to the equipment of modern smartphones with camera, accelerometer and a magnetic compass, we have several options for tracking at our disposal. Tracking based on sensors has received significant attention due to its simplicity. From the sensors alone the phone’s pose can be estimated, but since signals from the sensors are often noisy the resulting AR experience is often not optimal. However, sensors are not subject to any drift which is an important property we can leverage to make overall tracking more robust.

The other option for tracking is visual tracking using image features. This is much more accurate than sensor based tracking, however it can easily fail during rapid movement and in some cases it can be subject to drift. Our strategy is to combine sensor based tracking and visual tracking to keep the good bits of both methods. Throughout this section, we make the simplifying assumption that a tracked object’s movement is orthogonal to the cameras viewing direction, so that we can model an object’s pose merely using a translation and a rotation. This assumption holds true for most objects that are at a reasonable distance from the camera, which applies to most AR scenarios. Furthermore, for the placement of the label we even discard the rotational component of the movement, since we focus on text labels as augmentation, which should not change orientation to ensure good readability. Thus, ultimately we are only interested in the 2D screen location of a tracked object. We found that for our
purposes this simple model works very well in practice. In cases where the mo-
tion model is not able to capture an object’s pose correctly (e.g. during scale
change) the tracker can be reinitialized by sending a new request to the recog-
nition service. Note, that this also helps to overcome drift, i.e. re-initiating the
client-side tracker with a request to the recognition server increases robustness
even more.

In the following two sections we will first describe visual tracking, and imme-
diately after the integration of sensor tracking with the visual trackers.

5.4.1 Visual feature tracking

For tracking we use FAST [Rosten and Drummond 2006] corners in conjunc-
tion with $8 \times 8$ pixel image patches as feature descriptors. Whenever a recog-
nition request is sent to the server the extracted features of the current frame
are kept as reference features.

We make a distinction between two different modes of visual tracking, direct
tracking and incremental tracking. In direct tracking mode we attempt to match
the features of the current frame against the reference features. If this succeeds
then the recognized object can usually be very accurately located, with local-
ization error typically in the order of a single pixel. Direct tracking can fail for
a number of reasons, the most obvious of which is when the recognized object
moves outside the field of view of the camera. Even when a recognized object
is visible on the screen, direct tracking can fail since for objects very close to
the camera the assumption of Euclidean motion between frames might not hold
true and the simple image patch features we use are neither rotation nor scale
invariant.

In these situations incremental tracking jumps in, which matches the features of
the current frame against the features of the previous frame, in order to estimate
the inter-frame motion. The location of a tracked object is then updated solely
based on the incremental movements estimated between successive frames. Due to the limited accuracy of the feature detection ($\pm 0.5$ pixel) and image
noise an accumulation of small localization errors may happen, which means
incremental tracking is subject to drift. Thus, if the tracker remains in this
mode for several successive frames, a new recognition request is initiated.

In both tracking scenarios feature correspondences are calculated using simple
neighbor search with the $L_2$ norm on the patch descriptors. Several heuris-
tics are put in place to further speed-up the matching process. For instance we a-priori reject features with different FAST corner polarities and under the assumption that the inter-frame motion is small we can even further reduce the set of correspondence candidates.

5.4.2 Motion estimation

Once the basic feature correspondences between frames are estimated, for robust motion estimation we then use a procedure similar to what [Wagner et al. 2008a] refers to as “incremental tracking using pixel flow”. The idea is that each inter-frame correspondence yields a translation vector which is inserted into a two dimensional histogram. The total translation is then estimated by taking the weighted sum of all pixel flows in the neighborhood of the histogram’s primary mode. Since the motion model of [Wagner et al. 2008a] only consists of a pure translation without any rotational component, this procedure fails in the presence of rotation around the camera’s viewing direction. This is because no dominant mode can be found in the translation histogram. The situation is illustrated in Figure 5.5.

In order to account for translation and rotation we need to consider at least two feature correspondence pairs at the same time. We first take into account all 780 possible pairings of the 40 best matching features correspondences (i.e. the ones with smallest $L_2$ distance). We then reject any pair of correspondences where the distance of the two points in the reference frame and the current frame changes by more than 5 pixels. This is due to the assumption of planar Euclidean motion without scale change, where the distance between two matched features is expected to remain the same in every frame. Furthermore, if the two features in either the first or second frame are less than 5 pixels apart, then the correspondence pair is also rejected.

The remaining correspondence pairs are used for motion estimation. As in [Wagner et al. 2008a] we use a two dimensional histogram. However, instead of considering the translation of individual correspondences, we consider the translation and rotation of correspondence pairs. For each considered correspondence pair $((\vec{A}, \vec{A}'), (\vec{B}, \vec{B}'))$ we now first rotate the two points $\vec{A}$ and $\vec{B}$ in the reference frame around the labels position such that the difference vectors $\vec{d} = \vec{A} - \vec{B}$ in the reference frame and the difference vector $\vec{d'} = \vec{A}' - \vec{B}'$ in the current frame become co-linear. This is illustrated in Figure 5.4. If $\vec{A}''$ and
Figure 5.4: Before a translation estimate is calculated we compensate for any rotation around an object's label in the reference frame.

\( \vec{B}'' \) denote the rotated points in the reference frame, then a translation estimate for this feature pair is given by

\[
\Delta \vec{x} = \frac{\vec{A}' - \vec{A}''}{2} + \frac{\vec{B}' - \vec{B}''}{2}
\]

Each resulting estimate \( \Delta \vec{x} \) is then inserted into a two dimensional histogram. If the inter-frame motion can be properly approximated by only a translation and rotation, we expect to find one single mode in the histogram as demonstrated on the bottom right of Figure 5.5. Finally, we determine whether tracking across the two frames was successful by checking if the following conditions are met:

- At least 120 votes are cast
- The histogram bin with the highest number of votes together with its 8 neighboring bins must contain at least \( \frac{1}{4} \) of all cast votes
• The second highest mode of the histogram must either be very close to the first mode or contain less than half the amount of votes

If all these conditions are satisfied, then we assume that tracking has been successful and keep a weighted sum of all translation estimates in the neighborhood of the histogram’s primary mode as translation estimate. This voting scheme offers robustness against potential remaining outliers, as isolated votes will not affect the final result.

For each frame the visual tracker always first tries to perform direct tracking and uses the aforementioned criteria to determine if direct tracking was successful or not. If direct tracking fails, tracking switches to incremental mode, using the same criteria to determine if tracking was successful or not. If this also fails, we can still resort to sensor tracking, which is described in the next section.

### 5.4.3 Sensor tracking

We now want to combine our visual tracking with sensor tracking. Instead of trying to fuse the cues from both trackers, we use the input from the sensors as watchdog for the visual tracking system. As mentioned earlier, the pose of a mobile phone relative to the earth surface can in principle be fully determined from the sensors. When the camera parameters are also known, every camera pixel can be mapped to a point on a sphere with a fixed radius around the mobile phone\(^1\). Because the accelerometer and magnetic sensors are used to measure the earth’s gravitational and magnetic field directly, the resulting pose estimates are not subject to drift. However, the sensors built into consumer devices provide only very noisy signals and we also found that on Android phones the sensors’ outputs are provided at irregular time intervals. Thus in order to accurately place a label on the phone’s screen these signals need to undergo heavy filtering.

We let the visual and the sensor tracker run independently. The watchdog mechanism then works as follows. For every frame where direct visual tracking succeeded we assume that the tracked object has been accurately localized,

---

\(^1\)This is exploited in AR applications like Layar [lay] or Wikitude [wik] where based on the user’s position (obtained using GPS) markers for nearby objects like buildings or other landmarks are displayed on the mobile phone’s screen.
thus we update the sensor tracker’s view of the world, so that it matches the visual tracker. Whenever direct tracking fails and the visual tracker is in incremental tracking mode we compare the proposed label locations of the visual and sensor tracker. When over 10 successive frames the proposed label locations do not coincide, we assume that the incremental tracker suffered from drift and assume the tracked object has been lost. In this case we submit a new recognition request to the recognition service and reset the visual tracker to match the sensor tracker’s view of the world.
inter-frame motion with obvious rotational component

Figure 5.5: Bottom left: When considering the translation vectors of individual correspondences in the presence of a rotation, then no distinct translation direction can be determined. Bottom right: If however we compensate for the rotational component around an object label in the reference frame, then a clear consensus on the labels translation vector is found.
5.5 Implementation details

For the client-side implementation of our system we chose the Android mobile platform. This is due to the easy access to advanced features such as camera stream, GPS, sensor data etc. and the modern programming environment, which makes development more productive than on other mobile platforms. The user interface, network handling and sensor handling were implemented in Java. However, for the visual tracking we had to resort to native code in order to allow for interactive frame rates. Frames are passed from Java to native C code, which handles feature extraction, feature matching and motion estimation. Frame size is set at 480*320 pixels. With these settings we reach about 12 fps on a Google Nexus One phone. The Android platform is not yet fully optimized for vision based AR applications. There is a substantial overhead due to unnecessary memory allocation and garbage collection when grabbing frames from the camera. Additionally it is not trivial to synchronize the tracking results with the displayed frames resulting in a slightly offset label position.

5.6 Experiments and results

In order to perform reproducible tracking experiments we recorded video sequences together with sensor data, so that all experiments could be done offline. A groundtruth was generated by manually annotating every frame in the recorded videos. We found that even though sensor tracking is quite inaccurate with typical localization errors on the order of 50 pixels it’s very effective in detecting failures caused by drift during incremental tracking. To demonstrate this we recorded a special sequence, where a rapid movement cause incremental tracking to fail. We observed that within less than 2 seconds, the sensor tracker detects this failure and resets the visual tracker. Figure 5.6 shows an experiment for the visual part of the tracking pipeline. It can be observed, that for direct tracking the localization error remains bounded and reaches at most 4 pixels. For incremental tracking we found that even after 50 frames the accumulated error stays below 8 pixels.

Figure 5.7 shows frames from a sequence recorded under live operating conditions on a Google Nexus One. In the beginning the user points his phone at a book which is recognized by the kooaba [koo] recognition service. Next
5.7. Conclusions

We have demonstrated a fully functional and complete AR system which recognizes and tracks stationary as well as mobile objects. It combines a client-
side implementation on an Android powered mobile phone together with a server-side object recognition service. The client-side application allows for real-time tracking and interactive usage on state of the art smart phones. Our server-side provides an AR compatible object recognition service for 300000 object clusters with response times strictly under 2 seconds using a memory efficient geometric verification method. In our whole set-up we do not use any markers and do not require GPS information for object recognition and tracking.
Reciprocal Nearest Neighbours

6.1 Introduction

We are interested in retrieving images showing a particular object from a large database of reference images. This is an important problem with applications in Web image retrieval, mobile visual search, or auto-annotation of photos. Typically, object types covered by the images in the database consist of landmark buildings, scenery or other 3D objects.

Most approaches to solve this problem are based on the “visual words” concept, which in essence borrows techniques from text retrieval, after quantizing localized visual features into visual vocabularies [Nistér and Stewénius 2006, Philbin et al. 2007, Sivic and Zisserman 2003]. This method has turned out to be very powerful, since it allows for scalable retrieval in databases of millions of images at quite high precision. Even though astonishing progress has been made in terms of scalability and precision, accuracy on common retrieval benchmarks still shows room for significant improvements. And of course, any such accuracy improvement should ideally not affect memory consumption or retrieval time.

Thus, many recent works towards improving accuracy have focused on improving features, visual vocabularies or distance measures on a quite general level. Beyond these rather general measures, a further vantage point for improvement is given by exploiting the specific differences between text and visual retrieval. For instance, in images we can exploit the geometric arrangement of visual features (in 2D or 3D), whereas in text documents we have only access to the sequential arrangements of words in lines of text. Exploiting this geometric structure using RANSAC [Fischler and Bolles. 1981] or similar kinds of esti-
mations is consequently a very common step taken in order to improve retrieval accuracy [Philbin et al. 2007].

In this chapter we try to exploit another characteristic specific to visual data in order to improve accuracy of object retrieval results: often, the reference database contains many images showing the same object covering it from varying viewpoints etc. We make use of this by constructing a graph on the image database connecting each image with likely related images. At query time this graph is used to construct a set of database images that are closely related to the query image, then based on this close set the rest of the database is re-ranked. As we will show this has two benefits: first, treating the two sets with different similarity measures allows for compensation for the “curse of dimensionality”, i.e. the degradation of distance functions in high dimensional spaces. Second, it allows for dealing with the uneven distribution of images in the data space. Dealing with both challenges has very beneficial effect on retrieval accuracy.

The main contribution of this chapter is a method that improves image retrieval purely on the bag-of-words level. It does so without relying on lower-level information like for instance the geometric arrangement of features or the geometry of the descriptor space. As such our method can be used in a wide variety of settings. We also achieve very competitive results at reasonable overhead in memory usage and very little additional computational complexity during query time.

The remaining part of the chapter is structured as follows: We first discuss related work in the immediately following section. Section 6.3 lies the basis for our method, by discussing some key characteristics of visual words based object retrieval. We introduce our method for more accurate object retrieval in Section 6.4. Experiments and analysis of the effects of optimization on retrieval tasks follow in Section 6.5. Section 6.8 concludes the chapter.

### 6.2 Related work

Our work relates to recent contributions in the field of object retrieval with visual vocabularies in several aspects. The relevant works build on the common bag-of-features retrieval approach and have proposed improvements, which can be roughly grouped into three categories.

A first group of works deals with improvements on the feature level. In descriptor space the Euclidian distance is often used to assess the similarity of
features. However, it has been shown this is not the optimal similarity measure in most situations. In the context of large scale image retrieval this problem has recently been addressed by several works. For instance in [Mikulík et al. 2010] a probabilistic relationship between visual words is proposed as an alternative distance measure. It is based on an “oversegmentation” of the descriptor space with an extremely large vocabulary, and probabilistic relations between the visual words. This way, for each feature mapped to a visual word, a statistic of alternative visual words is learned. The relations are learned offline from a large set of feature tracks. Slightly similar is the work [Philbin et al. 2010], where data is used to learn a projection from SIFT feature space to a new Euclidean space, such that clustering is more likely to put matching descriptors into the same visual words.

A second group of works deals with the quantization artifacts introduced while assigning features to visual words. The most common effect of quantization artifacts is, that for two images showing the same object, corresponding features are not assigned to the same visual word. One way of dealing with this problem is by assigning each feature descriptor to multiple visual words as proposed in [Philbin et al. 2008], however the more words are assigned to a feature, the more posting lists in the inverted index have to be traversed, thus increasing the query time. In [Jegou et al. 2008], Jégou et al. addressed this problem by first constructing a relatively coarse vocabulary plus a binary signature for each feature. When a feature of the query images is assigned to a visual word of the coarse vocabulary, the binary signature is used to filter out database features by setting a threshold on the Hamming distance.

A third group of works deals with shortcomings on the document retrieval or database level. In [Chum et al. 2007], Chum et al. adopt query expansion (that originated in text retrieval) to the visual domain. Strict geometric verification is applied to the initial top list in order to extract a set of images that are very likely to be relevant to the query. Then a generative model is used to fuse the information provided by the additional images into a new query, which significantly increases recall.

A common cause of problems is due to the independence assumption between visual words, commonly used because of efficiency reasons. In reality this independence assumption is violated and some visual words co-occur more often than others. This can severely degrade retrieval accuracy. If for instance the query contains a set of frequently co-occurring visual words, then it is likely to match to unrelated images that contain the same set of co-occurring visual
words. These sets are commonly referred to as bursts [Jégou et al. 2009] or co-ocsets [Chum and Matas 2010]. In [Jégou et al. 2009], Jégou et al. evaluated several voting schemes that account for intra- and inter-image bursts. Chum et al. [Chum and Matas 2010] addressed this problem by finding and removing sets of frequently co-occurring visual words. In both cases improvement in retrieval accuracy was demonstrated. Also operating on the document vector level, Jégou et al. [Jégou et al. 2007, Jégou et al. 2010] improved the accuracy of visual word retrieval, by accounting for changes in the local distributions of the visual word vectors. To this extent, they introduced an iterative update scheme that modifies the distance function between vectors in a way that nearest neighborhood relationships become more symmetric.

Most similar to this chapter are probably [Chum et al. 2007] and [Jégou et al. 2010]. As we will explain in the following sections in more detail, the key differences to our work are that we do not rely on lower level information like for instance the geometric arrangement of features and we do not symmetrize nearest neighbor relationships (in contrast to [Jégou et al. 2010]).

6.3 Motivation

In this section we motivate our approach for improving accuracy of object retrieval by two key observations. Before we discuss the observations we give a brief overview of object retrieval with visual words.

Overview of object retrieval with visual words. In visual word based retrieval images are represented as sparse high dimensional visual word vectors. Given a query vector, visual search is formulated as ranking the vectors in the database according to their distance or similarity to the query vector. These vectors are constructed by first extracting a set of local features (usually SIFT [Lowe 1999] or SURF [Bay et al. 2006] features) for a given image which are then quantized using a visual vocabulary. The visual vocabulary is commonly learned by clustering a random sample of feature descriptors, where the number of cluster centers $K$ is usually somewhere around $10^6$. The quantization indices correspond to the non zero elements of the visual word vectors.

In the bag-of-words model, each non zero element of the vector counts how many times the visual word appears in the image. Since the number of visual
words in an image is usually many orders of magnitude below the size of the visual vocabulary, the visual word vectors are extremely sparse. Using an inverted index this sparsity is exploited to efficiently calculate the similarity of a query vector to all database vectors.

For all experiments in this chapter we use the same similarity function as [Jegou et al. 2008], which corresponds to the bag-of-words model with an additional inverse document frequency weighting term:

\[
\text{sim} (q, d) = \frac{\sum_{i=1}^{K} q_i d_i \text{idf}(i)^2}{\|q\| \|d\|} \tag{6.1}
\]

\[
\text{idf} (i) = \log \left( \frac{\sum_{i=1}^{K} \sum_{d \in D} d_i}{\sum_{d \in D} d_i} \right) \tag{6.2}
\]

where \(q\) and \(d\) are visual word vectors of length \(K\) and \(D = \{d^1 \ldots d^N\}\) is the set of database vectors.

**Observation 1: Similarly functions degrade quickly in high dimensional spaces.** A fundamental issue for visual word based image retrieval is the high dimensionality of the visual word vector space. While this high dimensionality facilitates fast search, it also has the effect, that most distance or similarity measures quickly degenerate at points far away from the query vector.
An illustration of this phenomenon is shown in Figure 6.1 where we plot the similarity measure $\text{sim}(q, d)$ (cf. Equation 6.1) for the top 70 ranked images in the Oxford5k data set [oxf] for a given query. Correctly retrieved matches from the evaluation data set are denoted by a red circle. Most images with high similarity are of course true positives, however the similarity curve quickly flattens out giving relevant and non relevant images almost the same score at lower ranks. So the similarity measure is very useful for images close to the query, but it loses its utility far away from the query. One way of dealing with this problem is by modifying the similarity measure $\text{sim}(q, d)$ in a way that more relevant images are pushed closer to the query and irrelevant images are pushed away from the query. For instance, Hamming embedding [Jegou et al. 2008] or soft visual word assignment [Philbin et al. 2008] do this by reducing quantization artifacts.

Descriptor space learning techniques [Philbin et al. 2010, Mikulík et al. 2010] push relevant images closer to the query by correcting for the fact that the Euclidean norm is not a perfect distance measure in SIFT or SURF descriptor spaces. In addition, filtering irrelevant images from the ranked lists is typically achieved by geometric verification [Philbin et al. 2007] or similar methods (e.g. [Jegou et al. 2008]).

In this chapter we try to address these effects of the curse of dimensionality in a slightly different way. Accepting that the similarity measure can degrade quickly, we will split the database vectors at query time into two groups. One group which is close to the query, and for which the regular similarity measure $\text{sim}(q, d)$ can still correctly separate relevant from non-relevant vectors, and a second group, for which the similarity measure can not distinguish between relevant from non-relevant vectors anymore. For the second group we use a different similarity measure.

**Observation 2: Non-reciprocity of near neighbor relationships.** Let us define the $k$-nearest neighbors (i.e. the top-k list) of a vector $q$ as the $k$ most similar vectors in the database $D$:

$$\text{top}(k, q) \subset D$$

$$|\text{top}(k, q)| = k$$

$$\text{sim}(q, a) > \text{sim}(q, b) \quad \forall \ a \in \text{top}(k, q) \quad b \in D \setminus \text{top}(k, q)$$
While the similarity measure $\text{sim}(q, d) = \text{sim}(d, q)$ itself is symmetric, nearest neighbor relationships are not. This means that $a \in \text{top}(k, b)$ does not imply $b \in \text{top}(k, a)$ in general.

We define the set of $k$-reciprocal nearest neighbors $R(k, a)$ of $a$ as

$$R(k, a) = \{ b \in \text{top}(k, a) \land a \in \text{top}(k, b) \}$$

which is of course trivially symmetric. The $k$-reciprocal nearest neighborhood relationship $b \in R(k, a)$ is also a much stronger indicator of similarity than the unidirectional nearest neighborhood relationship $b \in \text{top}(k, a)$, since it takes into account the local densities of vectors around $a$ and $b$.

We illustrate in Figure 6.2 the difference between the unidirectional nearest neighbor set $\text{top}(2, q)$ and the 2-reciprocal nearest neighbor set $R(2, q)$. $\text{top}(2, q)$ contains the node $a$ and $d$, $R(2, q)$ only contains node $d$, even though $a$ and $d$ are at the same distance from the query $q$. In such a situation it makes sense to assume that $d$ is more relevant to the query $q$ than $a$, since $a$ has a high similarity to other nodes that share no connection to $q$.

We are of course not the first to make this observation. Contextual dissimilarity measures [Jégou et al. 2007, Jégou et al. 2010] for instance are based on exactly this idea. However unlike [Jégou et al. 2007, Jégou et al. 2010], we do not directly symmetrize nearest neighborhood relationships in this work. Instead we use $k$-reciprocal nearest neighbors as a tool to find images which are very likely to be related and to disambiguate database vectors that are far away from the query vector.
These two observations are the basis for our object retrieval method, which will be discussed in the following section.

6.4 Our Approach

At query time we want to separate the database into two disjoint sets, the close set which contains images highly relevant to the query and the far set which simply refers to the rest of the database. The final ranking list is the concatenation of the close set for which parts internally are ranked according to the original similarity measure $\text{sim}(q, d)$ (cf. Equation 6.1) and the far set which is ranked according do a different similarity measure. We first discuss how the close set is constructed and then describe the similarity measure that is used for the far set.

6.4.1 Close set construction

In order to identify images highly related to the initial query image $q$, we start by adding the $k$-reciprocal nearest neighbors $R(k, q)$ of the query to the close set.

In Figure 6.3 we show for a query in the Oxford5k data set how precision and recall of $R(k, q)$ change for various values of $k$. With higher values of $k$, recall is increased and saturates while precision rarely decreases. Since in practice some images have very few $k$-reciprocal nearest neighbors, even for very large $k$, we grow the initial close set $N_{q,t=0}$ by iteratively adding neighboring nodes to increase recall. Nodes are only added if a set of conditions are met which are designed in a way, that only images that are very likely to be related to the query image are added.

We first define the forward rank $f$-rank$(a, q)$ of $a$ as the position that $a$ has in the top list of $q$ and the backward rank $b$-rank$(a, q)$ is defined as the position that $q$ occupies in the top-$k$ list of $a$:

\[
\begin{align*}
  f\text{-rank}(a, q) &= k & \iff & a \in \text{top}(k, q) \setminus \text{top}(k-1, q) \\
  b\text{-rank}(a, q) &= f\text{-rank}(q, a) \\
  a \in R(k, q) & \iff f\text{-rank}(a, q) < k \land b\text{-rank}(a, q) < k
\end{align*}
\]
6.4. Our Approach

Since we are only interested in finding nodes close to the query, we only consider nodes \( d \in D \) if their forward and backward rank relative to the query \( q \) do not exceed a certain threshold \( k_{\text{max}} \):

\[
f\text{-}\text{rank}(q,d) < k_{\text{max}} \quad \lor \quad b\text{-}\text{rank}(q,d) < \frac{1}{2} k_{\text{max}}
\]  

(6.10)

Ignoring all nodes which do not satisfy these constraints we grow \( N_{q,t=0} \) by the following procedure as described in Algorithm 1.

**Algorithm 1**: Expansion step

1. for \( t \leftarrow 0 \) to \( 2 \) do
2. \( N_{q,t+1} \leftarrow N_{q,t} \)
3. foreach \( n \in N_{q,t} \) do
4. if \( |N_{q,t} \cap R(k,n)| > \frac{1}{2} |N_{q,t}| \) then
5. \( N_{q,t+1} \leftarrow R(k,n) \cup N_{q,t+1} \)
6. if \( |N_{q,t} \cap R(k,n)| > |R(k,n) \setminus N_{q,t}| \) then
7. \( N_{q,t+1} \leftarrow R(k,n) \cup N_{q,t+1} \)

The first condition allows only nodes which are connected to at least half of the close set to bring in their neighbors. This high connectivity ensures that added nodes are very likely to be relevant to the query. The second condition relaxes this restriction slightly by allowing weakly connected nodes to bring in their neighbors if the amount of new neighbors is smaller than the amount of
Figure 6.4: Overview over the expansion rules. For the new set $N_{q,t+1}$ only nodes are considered which either occur in the first half of the top-$k$ list of the query image ($q \rightarrow n_3$), or if the query image occurs within the top list of the image ($n_1 \rightarrow q$).

Connections already made to the close set. Nodes added to $N_{q,t+1}$ are sorted according to $\text{sim}(q, d)$ and inserted in this order into the final close set. This procedure can be seen as a form of query expansion, however unlike [Chum et al. 2007] we do not rely on any geometric or other lower-level information. Figure 6.4 gives an overview of the two conditions for better visualization and in Figure 6.5 we give a real world example of the growing procedure.

In order to efficiently construct the close set for a new query, we pre-compute a directed graph $(d_1 \ldots d_n \in D, (u, v)_i \in V)$ on top of the image database. In this graph every node represents an image and a connection $(u, v) \in V$ from node $u$ to another node $v$ is made if node $v$ appears in the truncated $\text{top}(k_{\text{max}}, \cdot)$ list of $u$:

$$(u, v) \in V \iff v \in \text{top}(k_{\text{max}}, u)$$ (6.11)
where we used $k_{\text{max}} = 1000$ for all experiments in this chapter. Using Equations 6.8 and 6.9, $k$-reciprocal neighborhoods $R(k, q)$ can be efficiently determined.

In order to construct this graph, we query our retrieval system with every image in the database. While this step is quadratic in the number of images, we do not see this yet as a fundamental restriction since computation is trivially distributable. Also the query operation is quite fast, for a set of one million images we could compute the aforementioned graph in less than 5 hours using only 8 machines. It is reasonable to assume, that calculating the graph for 10 million images would still be feasible. For larger data sets, approximations may be used like for instance min hash [Chum et al. 2008] which has only linear complexity in the number of images. However since we can still deal quite comfortably with up to one million images we have not evaluated this for the purpose of our method. At query time the similarity of the query image to all database images is calculated and the graph is updated to include a node for the query image.

### 6.4.2 Far set re-ranking

Once the close set is constructed, it is used to re-rank the rest of the database. Since images outside of the close set are likely to have a low similarity to the query, the original similarity measure $\text{sim}(q, d)$ is not useful anymore. From the vantage point of the query, images outside of the close set all look equally dissimilar. However if we turn the tables, and look from the position of an element in the far set this might not be true.
Images in the *far set* which are closely surrounded by other images in the *far set* will populate their top($k$, ·) list with their close neighbors but not the ones from the *close set*. However images which are dissimilar to the entire database but still rather close to the initial query can populate their top($k$, ·) list with images from the *close set*.

Intuitively it also makes sense that images which do not have any close neighbors except for images in the *close set* are more likely to be relevant to the query, than images that have close neighbors which are not related to the *close set*. In order to make use of this contextual similarity we calculate for each document in the *far set* ($f \in D \setminus N_{q,3}$) the average rank that images in the *close set* would have if image $f$ were used as a query:

$$\text{sim}(f, q) = \text{cutoff} - \frac{1}{|N_{q,3}|} \sum_{c \in N_{q,3}} \min(b\text{-rank}(f, c), \text{cutoff})$$

where we use a cutoff to account for the fact that only a truncated version of the ranking lists is present at retrieval time. We used cutoff $= 3000$ for all experiments in this chapter.

### 6.5 Experiments

In this section we evaluate the performance of our method on five different datasets. First we give an overview of the datasets. Then we assess the performance of the *close set* and the performance of the *far set* re-ranking separately. At the end a comparison of our full method to the baseline approach is given.

#### 6.5.1 Evaluated datasets

We evaluated our method on the Oxford5k [Philbin et al. 2007, oxf], Oxford105k [Philbin et al. 2007], Paris [Philbin et al. 2008, par], University of Kentucky [Nistér and Stewénius 2006, nis] and the INRIA Holidays [Jegou et al. 2008, inr] dataset. Oxford5k and Paris are relatively small datasets containing 5063 and 6412 database images each. For both datasets 55 queries with ground truth are provided. In the Oxford dataset on average 10% of the database is relevant to a query, whereas for the Paris dataset almost 30% of
6.5. EXPERIMENTS

images are relevant. Furthermore, there is a high variance in the number of ground truth images for the different queries. The Kentucky dataset consists of 10 200 images for which always 4 show the same object. The Holidays dataset consists one million distractor images and 1 491 relevant images of which 500 are queries. For a large portion of the queries in this dataset there are only 1 or 2 relevant images.

The Oxford105k dataset consists of Oxford5k and 100 000 distractor images. As we did not have access to the original distractor images we downloaded 100 000 random geotagged images from Flickr\(^1\) and Panoramio\(^2\) which have been taken at least 500 km away from Oxford in order not to incorporate possibly relevant images that could artificially pollute the results. Furthermore we ensured that all downloaded images have resolutions between 768 × 1024 and 1024 × 1024 pixels, as in the original Oxford5k dataset.

We used Hessian Affine SIFT descriptors and approximate \(k\)-means [Philbin et al. 2007] to cluster a visual vocabulary with 500 000 centroids for Oxford5k, Paris and Kentucky each. For Oxford105k we used the same vocabulary as for Oxford5k. For the Holidays dataset we received the pre-calculated visual words for a 200\(k\) visual vocabulary from the authors of [Jegou et al. 2008]. Thus our baseline and the one from [Jegou et al. 2008] are exactly the same.

As performance measure, we used mean average precision (\(mAP\)) on the Oxford5k, Oxford105k, Paris and Holidays dataset while for the University of Kentucky dataset we use the top-4 score as defined by [Nistér and Stewénius 2006].

6.5.2 Close set accuracy

In the first part we demonstrate, that the construction of the close set which forms the first part of the final ranking list leads to higher accuracy than simply taking the top-\(k\) elements of the original ranking list. The size of the close set for a given query is dependent on the number of similar images in the database. Queries with many similar images in the database have a larger close set than queries with only few similar images in the database. Furthermore, the size of the close set depends on the threshold \(k\). By varying \(k\) we produce close sets of different sizes. The far set for which in this experiment regular ranking (\(1\)http://www.flickr.com
\(2\)http://www.panoramio.com
As can be seen by the blue lines in Figures 6.6,6.7,6.8,6.9,6.10 this gives a major improvement on all datasets for a wide range of $k$. The mAP and top-4 score give high importance to the first part of a ranking list, which is exactly where the close sets increases accuracy.

### 6.5.3 Far set accuracy

We investigate the effect of replacing the close set by simply a truncated top-$k$ list for different values of $k$ and re-rank the far set using this list according to Equation 6.12.

As can be seen by the green lines in Figures 6.6,6.7,6.8,6.9,6.10 this gives an improvement on all datasets for small $k$. However as $k$ increases the performance asymptotically degrades back to the base line, since larger and larger portions of the beginning of the final ranking list are ranked using the same similarity measure as the baseline. This is especially visible for the Kentucky dataset. Since the top-4 score only considers the first 4 positions of the ranking list, for $k > 4$ the performance is equal to the baseline.

### 6.5.4 Full method

The full method combines the aforementioned rank list construction methods, such that the first entries of the final ranking list consist of the close set to which the far set is appended.

The red line in Figures 6.6,6.7,6.8,6.9,6.10 show the final result of the whole method for different thresholds $k$. The combination of close set construction and far set re-ranking leads to superior results over the baseline in all cases.

Figure 6.11 shows the average precision for the baseline versus the average precision of our improved method for individual query images for a fixed $k = \arg\max_k \text{mAP}(k)$. Off-diagonal markers in the upper left triangle show a performance improvement, markers in the lower right triangle a degradation. For the University of Kentucky dataset a slightly different visualisation approach was taken. The top-4 score of the baseline method is plotted against the
top-4 score of our new method. Each of the bubble’s area corresponds to the number of images at this coordinate.

The combination of both methods yields in all datasets to superior results over the baseline. As the performance decays slowly, $k$ is not as dataset specific as it might seem. Setting it to somewhere between 20 and 40 gives good results for real world datasets used in image retrieval applications.
As can be seen in Table 6.2 for Oxford5k, Oxford105k and Paris we compete with the state of the art, however we do so without exploiting lower level information. For the Kentucky dataset we miss the state of the art only by 0.01 of top-4 precision. For the Holidays dataset it is well known that Hamming Embedding and Weak Geometric Consistency Constraint can greatly improve results, further more the 200$k$ visual vocabulary is quite small for such a large dataset.
We chose to evaluate our method on this challenging dataset to demonstrate that even under very unfavorable conditions we achieve a significant improvement.

As in recent years, many advances have been made over standard bag-of-words retrieval and reported results usually only make use of a subset of methods that are known to improve results. For instance in [Philbin et al. 2010] a projection from SIFT feature space to a new Euclidean space is learnt, such that matching descriptors are more likely to be assigned to the same visual word. The reported results are with spatial reranking on the top 200 returned images, however no query expansion is performed. Other works like for instance [Mikulík et al. 2010] report results with query expansion and descriptor space learning methods. Thus retrieval performance measures like mAP always have to be seen in context when a new method is introduced. In Table 6.2 we tried to give this context for the results reported by us and other authors.

Table 6.1: Additional memory overhead per dataset and average time overhead per query.
Table 6.1 shows an overview over the total memory overhead per dataset and the average query time overhead for each query. As for each image in the database the forward- and the backward ranking lists need to be stored, the memory overhead grows linearly with the database size. This overhead is in the same order of magnitude as for instance Hamming Embedding [Jegou et al. 2008]. The query time overhead is mainly dependent on the length of the backward ranking list and the chosen threshold $k$ as this restricts the size of the close and far set.
6.5. EXPERIMENTS

Table 6.2: mAP for different data sets compared to results of state of the art results. In [Mikulík et al. 2010] Mikulík et al. did not specify whether or not they added 1 Mio distractor images to the INRIA dataset. We assume they did not. Furthermore they manually corrected the orientation (sky-is-up) for images in the dataset. The result from [Jégou et al. 2009] Jégou et al. was extracted from a plot, since they did not report it in numerical form.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Baseline max(\frac{mAP}{top-1})</th>
<th>Our method mean (\pm 3 \times \text{std}) for (k \in [20 \ldots 40])</th>
<th>[Jégou et al. 2010]</th>
<th>[Jégou et al. 2009]</th>
<th>[Mikulík et al. 2010]</th>
<th>[Chum et al. 2007]</th>
<th>[Chum and Matas 2010]</th>
<th>[Philbin et al. 2010]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oxford5k</td>
<td>0.674</td>
<td>0.814 ((k = 18))</td>
<td>0.785 (\pm 0.018)</td>
<td>0.685</td>
<td>0.849</td>
<td>0.795</td>
<td>0.782</td>
<td>0.864</td>
</tr>
<tr>
<td>Oxford105k</td>
<td>0.567</td>
<td>0.767 ((k = 37))</td>
<td>0.757 (\pm 0.018)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paris</td>
<td>0.693</td>
<td>0.803 ((k = 36))</td>
<td>0.798 (\pm 0.009)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INRIA</td>
<td></td>
<td></td>
<td></td>
<td>0.848</td>
<td></td>
<td>0.758</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INRIA +1 Mio</td>
<td>0.315</td>
<td>0.423 ((k = 40))</td>
<td>0.419 (\pm 0.006)</td>
<td>0.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kentucky</td>
<td>3.5</td>
<td>3.67 ((k = 4))</td>
<td>3.528 (\pm 0.021)</td>
<td>3.68</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spatial Verification</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Query Expansion</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Assignment</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Descriptor Space Learning</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vocabulary Size</td>
<td>500 K</td>
<td>500 K</td>
<td>500 K</td>
<td>19 (\times) 30 K</td>
<td>20 K</td>
<td>16 Mio</td>
<td>1 Mio</td>
<td>1 Mio</td>
</tr>
<tr>
<td>(INRIA 200 K)</td>
<td>(INRIA 200 K)</td>
<td>(INRIA 200 K)</td>
<td>(Rank Aggregation)</td>
<td>(Rank Aggregation)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+64-bit-HE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.6 Close Set reranking examples

We briefly illustrate how the close set reranking affects the original top lists of the Oxford5k dataset. We choose queries for which our method either performed very well or very poorly. Figures 6.13, 6.12, 6.14, 6.15 and 6.16, show top-50 ranking lists generated from our method (left side) and compare them to standard bag-of-words retrieval (right side). Figures 6.13, 6.12 and 6.14 are the 3 queries that had the biggest improvement in average precision over the baseline, whereas Figures 6.15 and 6.16 are the queries that had the biggest decrease in average precision. We added colored bars under each image to indicate if the image belongs to the close set and during which expansion step it is added:

- red corresponds to $N_{q,0}$
- cyan corresponds to $N_{q,1}$
- yellow corresponds to $N_{q,2}$
- blue corresponds to $N_{q,3}$
Figure 6.12: Results for bodleian_5_query, old AP was 0.33 our method increased the AP to 0.97
Figure 6.13: Results for bodleian_1_query, old AP was 0.24 our method increased the AP to 0.82
Figure 6.14: Results for ashmolean_1_query, old AP was 0.46 our method increased the AP to 0.90
Figure 6.15: Results for christ_church_4, old AP was 0.87 our method decreased the AP to 0.80
Figure 6.16: Results for christ_church_1, old AP was 0.84 our method decreased the AP to 0.73
6.7 Efficient representation of the neighbourhood graph

As our method adds a non negligible storage overhead, we investigate some possibilities to reduce the amount of overhead. In short our method requires a neighbourhood graph to be stored, with a fixed amount of neighbours $k$ for each of the $n$ documents. For each of the $k$ neighbours two values are stored, the the neighbour id and the distance to the neighbour. A straightforward implementation would use a 32 bit integer for the neighbour id and a 32 bit floating point value for the distance. This gives in total an overhead of $n \times k \times 8$ bytes, which while linear in the number of documents $n$ and neighbourhood size $k$ is still substantial. First we argue, that the 32 bit integers for the neighbourhood ids can only be moderately compressed and for larger and larger databases ($\lim_{n \to +\infty}$) no lossless compression is possible anymore. For simplicity we assume, that neighbourhood ids are stored in a fixed order (e.g. according to their distance) and that $k$ integers in the range $[0..n-1]$ can be represented by $k \log_2 n$ bits. There are $\binom{n}{k}$ possibilities of choosing $k$ documents out of the entire database of $n$ documents and there are $k!$ possibilities of ordering these documents. So the total number of possible neighbourhood ids in a fixed ordering is given by Equation 6.13.

$$\binom{n}{k} \times k! = \frac{n!}{k!(n-k)!} \times k! = \frac{n!}{(n-k)!}$$ (6.13)

Therefore the minimum number of bits required to store the ids is

$$\log_2 \left( \frac{n!}{(n-k)!} \right)$$ (6.14)

which is less than a naive encoding $k \log_2 n$ since

$$\frac{n!}{(n-k)!} = n(n-1) \ldots (n-k+1) \leq n^k$$ (6.15)

$$\Rightarrow$$

$$\log_2 \left( \frac{n!}{(n-k)!} \right) \leq k \log_2 n$$ (6.16)
However in the limit $\lim_{n \to +\infty}$ the difference vanished:

$$\lim_{n \to +\infty} \log_2 \left( \frac{n!}{(n-k)!} \right) - k \log_2 n = 0 \quad (6.17)$$

So assuming that the neighbourhood ids in the graph are independently distributed (which is not quite true), then the naive encoding does still quite well compared to the optimal encoding.

While there seems no easy way to reduce the number of bits spent on the neighbourhood ids, there is a fairly simple way to greatly reduce the number of bits required to store the neighbourhood distances. We can make use of the fact that the neighbours for each node are ordered according to their distance, as the ordered distances can be seen as a monotonic decreasing function. This function can be well modeled for instance using simple function approximation methods. In Figure 6.17 on the top we plot the achieved mAP on the Oxford 5k dataset vs approximation error for different error thresholds. In this experiment 3000 neighbours are computed for every document and the $k$ parameter is set to 10 (cf. Equation 6.6). Each of the neighbourhood lists is approximated via piecewise linear approximation. The distances (or similarities in this case) are normalized, such that the similarity between two equal images is one. Different error thresholds are set such that the linear piecewise approximation is guaranteed to be below the given error threshold for every point.

The lower part of Figure 6.17 shows the average required bytes per document to represent the approximated neighbourhood distances, one can clearly see, that as little as 70 bytes per image suffice to represent neighbourhood distances accurate enough that there is no loss in mAP. This compares very favorably to the naive storage of all neighbourhood distances using 32bit floating point values, which in this experiment would amount to 12 kilo bytes of storage per image.
6.8 Conclusion

We have demonstrated that a significant improvement in bag-of-words retrieval can be achieved, without considering the geometric arrangement of features in an image nor by modifying the feature quantization step. Our method uses $k$-reciprocal nearest neighbors to identify an initial set of highly relevant images in the database which are then used to re-rank the remaining part of the database. On many data sets our approach competes with the state of the art. The memory overhead of our method is linear in the number of documents while the average query time overhead is neglectable.
7

Engineering Aspects

7.1 Features

In this section we demonstrate, that the choice of local features can have a significant effect on retrieval performance. We evaluate several popular local feature detectors and descriptors in great detail using the Oxford5k [Philbin et al. 2007], Paris [Philbin et al. 2008], Kentucky [Nistér and Stewénius 2006] and Holidays [Jegou et al. 2008] dataset. For simplicity the evaluation protocol is always the same. Features are extracted from all images of a given dataset and then a 500K visual vocabulary is generated using k-means with a best bin first modified kd-tree like in [Philbin et al. 2007]. The final evaluation measures follow the original publications. That means for the Paris and Oxford5k mAP is calculated, while for the Kentucky dataset the top-4 score is used. In this section no additional processing is considered like for instance Query Expansion [Chum et al. 2007, Chum et al. 2011] or Soft Matching [Philbin et al. 2008]. However we do consider a generalized form of local features where each feature vector is rescaled using an exponent on each entry as well as different similarity function in document space.

7.1.1 Binaries

Before we consider different features, we first take a look at several publicly available binaries that provide Hessian Affine feature detection [Mikolajczyk et al. 2005] and SIFT [Lowe 1999, Lowe 2004] feature description, as this is a very popular combination for image retrieval. While ideally one should not expect vast differences in the different implementations, we show that the reality could not be more different. We consider the following feature binaries:
• **HAFFINE** or *h_affine.ln* is a hessian affine region detector. It can not compute feature vectors by itself. It’s SHA1 is bb0cc10330a1af8478d0f10861f97db1485b6c0e and it was downloaded from [http://www.robots.ox.ac.uk/](http://www.robots.ox.ac.uk/)

• **EXTRACT1** or *extract_features.ln* is a region detector and descriptor. While this binary is able to process input regions from other binaries, it can not produce feature regions with no descriptor selected. We therefor only evaluate this binary only as a feature descriptor and standalone feature detector/descriptor. It’s SHA1 is bf14e6ec58b0214f4a397f1605647a9c28a16f1 and it was downloaded from [http://www.robots.ox.ac.uk/](http://www.robots.ox.ac.uk/)

• **EXTRACT2** or also *extract_features.ln* is supposedly a bug-fixed version of **EXTRACT1**. It’s SHA1 is b938be6c882372f4dc8bcb6403094868d3734513 and it was downloaded from [http://www.robots.ox.ac.uk/](http://www.robots.ox.ac.uk/)

• **COMPUTEDESC1** or *compute_descriptors.ln* is only region descriptor. Its SHA1 is 293d7bf4f2b5d4b33903081c1deef04b9865ca41 and it was downloaded from [http://www.robots.ox.ac.uk/](http://www.robots.ox.ac.uk/)

• **COMPUTEDESC2** is a feature detector and descriptor. Its SHA1 is 6d7549c34b2020e8c808ed0cc789e42d5620a6167 and it was downloaded from [http://www.irisa.fr/](http://www.irisa.fr/)

• **COMPUTEDESC3** is a different version of **COMPUTEDESC2**. Probably due to a bug, it can not handle region input points from other binaries. Its SHA1 is 89e808d9ba7433cc594406b1365c49d623cb42a and it was downloaded from [http://pascal.inrialpes.fr/](http://pascal.inrialpes.fr/)

• **COMPUTEDESC4** is a different version of **COMPUTEDESC2**. Its SHA1 is a103d8e1d675667847e43499cb6aadce9314de4d and it was downloaded from [http://kahlan.eps.surrey.ac.uk/](http://kahlan.eps.surrey.ac.uk/)
Table 7.1: mAP results for various detector (top to bottom) and descriptor (left to right) binary combinations on the Oxford5k dataset.

<table>
<thead>
<tr>
<th>Detector</th>
<th>EXTRACT1</th>
<th>EXTRACT2</th>
<th>COMPUTEDESC1</th>
<th>COMPUTEDESC2</th>
<th>COMPUTEDESC3</th>
<th>COMPUTEDESC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAFFINE</td>
<td>0.669</td>
<td>0.444</td>
<td>0.679</td>
<td>0.439</td>
<td>0.478</td>
<td></td>
</tr>
<tr>
<td>EXTRACT1</td>
<td>0.529</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTRACT2</td>
<td>0.115</td>
<td>0.563</td>
<td>0.202</td>
<td>0.609</td>
<td>0.646</td>
<td></td>
</tr>
<tr>
<td>COMPUTEDESC2</td>
<td>0.401</td>
<td>0.520</td>
<td>0.375</td>
<td>0.529</td>
<td>0.507</td>
<td></td>
</tr>
<tr>
<td>COMPUTEDESC3</td>
<td>0.412</td>
<td>0.512</td>
<td>0.396</td>
<td>0.522</td>
<td>0.523</td>
<td>0.518</td>
</tr>
<tr>
<td>COMPUTEDESC4</td>
<td>0.062</td>
<td>0.628</td>
<td>0.050</td>
<td>0.634</td>
<td>0.638</td>
<td></td>
</tr>
</tbody>
</table>

In table 7.1 we show mAP results on the Oxford5k dataset for various combinations of detector and descriptor binaries. The detector binaries generate Hessain Affine interest points and store the location and ellipse parameters in a human readable ASCII file, which is then passed to the descriptor binary. Whenever the descriptor and detector binaries coincide, we do not take the detour via the ASCII file, but rather let the binary output feature vectors directly. Since some descriptor binaries sporadically take an unreasonable amount of processing time (most likely due to a bug or poor implementation), we limited the execution time for feature extraction to 60 seconds per image. If feature description is not completed in this time, then we simply assume that the image contains zero features. We found that this step was not necessary for region detection binaries, as they ran fast enough. One can clearly see that many binaries do not work well together. Unfortunately we can only speculate on the possible reasons, as we did not have access to the source code of the binaries. Possible reasons could be, that the input format differs from the documented format. For the EXTRACT1 and COMPUTEDESC1 binaries we especially found, that they often took more than 60 seconds to execute in combination with certain detector binaries, however they produce excellent results in combination with the HAFFINE detection binary. Oddly enough tough EXTRACT2 which supposedly is an improved version of EXTRACT1 does not produce good results.
Figure 7.1: The top left shows a test image, the other images show extracted interest regions for different detector binaries.

Indeed some evidence for this is shown in Figure 7.1, where we plotted the extracted ellipses for a test image. One can clearly see, that different binaries return ellipses at different scales. As it is common practice to scale ellipses generated by a interest point detector before the feature descriptor is computed, the different result ellipse scales can be attributed to different default scaling factors. Additionally the feature detector binaries might expect the input ellipses to already be scaled to a certain size, thus when the scale of an interest point detector does not match the expected scale of a feature description binary, things can go horribly wrong. This may also explain why certain feature description binaries take an unreasonable amount of time to produce feature vectors, as the input regions by far larger than expected. One possibility to address this issue would be to look at the statistics of the produced ellipses in order to figure out what inherent scaling factor is added for each interest point detector binary. However we feel that such a statistical reverse engineering of the binaries goes well beyond the scope of this chapter.
Table 7.2: mAP results for various binaries that handle detection as well as description on the Oxford5k dataset.

<table>
<thead>
<tr>
<th></th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTRACT1</td>
<td>0.529</td>
</tr>
<tr>
<td>EXTRACT2</td>
<td>0.563</td>
</tr>
<tr>
<td>COMPUTEDESC2</td>
<td>0.529</td>
</tr>
<tr>
<td>COMPUTEDESC3</td>
<td>0.523</td>
</tr>
<tr>
<td>COMPUTEDESC4</td>
<td><strong>0.638</strong></td>
</tr>
</tbody>
</table>

Table 7.2 shows mAP values for all evaluated binaries which handle region detection and feature description together. It is safe to assume that the handover of detection data to the description step is done correctly in these binaries. Nevertheless we can still find drastic differences in the resulting mAP scores. These differences can come either from bugs in the implementation or the detector default parameters differ between the various binaries. Since we find that COMPUTEDESC4 by far produces the best results, we will use it for the remainder of this chapter. In Table 7.3 we compare several interest point detectors provided by the COMPUTEDESC4 binary. For completeness sake we additionally we give results for the OpenCV implementation of SIFT and SURF as well as the official SURF implementation. One can clearly see, that the Hessian/Harris-Laplace/Affine region detectors give the best results. Interestingly enough, the Hessian Affine region detector which is a very popular choice for large scale image retrieval is not the best, but still a good choice.
<table>
<thead>
<tr>
<th></th>
<th>oxford</th>
<th>paris</th>
<th>kentucky</th>
<th>holidays</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCV SURF</td>
<td>0.517</td>
<td>0.594</td>
<td>2.938</td>
<td>0.736</td>
</tr>
<tr>
<td>MSER</td>
<td>0.555</td>
<td>0.605</td>
<td>2.977</td>
<td>0.738</td>
</tr>
<tr>
<td>OpenCV SIFT</td>
<td>0.588</td>
<td>0.656</td>
<td>2.944</td>
<td>0.716</td>
</tr>
<tr>
<td>BIWI SURF</td>
<td>0.555</td>
<td>0.608</td>
<td>3.135</td>
<td>0.778</td>
</tr>
<tr>
<td>Hessian Affine</td>
<td>0.638</td>
<td>0.656</td>
<td>3.379</td>
<td>0.835</td>
</tr>
<tr>
<td>Hessian Laplace</td>
<td>0.640</td>
<td>0.662</td>
<td>3.407</td>
<td>0.778</td>
</tr>
<tr>
<td>Harris Affine</td>
<td>0.669</td>
<td>0.714</td>
<td>3.389</td>
<td>0.812</td>
</tr>
<tr>
<td>Harris Laplace</td>
<td>0.673</td>
<td>0.725</td>
<td>3.401</td>
<td>0.822</td>
</tr>
</tbody>
</table>

Table 7.3: mAP results on Oxford5k, Paris, Holidays and Top-4 scores on Kentucky using different region detectors and descriptors.

7.2 Feature Scaling

As the Hessian/Harris-Laplace/Affine SIFT features produced by the COMPUTEDESC4 seem superior to all other feature type, the next section will only focus on these 4 feature types. We investigate how an exponential scaling of the features affects retrieval performance. More precisely let \( \vec{f} = (f_1, \ldots, f_D) \) be a \( D \) dimensional feature vector, then we define the \( \alpha \)-exponentially scaled version \( \text{scale}(\vec{f}, \alpha) \) of \( \vec{f} \) as:

\[
\text{scale}(\vec{f}, \alpha)_i = \|\vec{f}\| \frac{\text{sign}(f_i)|f_i|^\alpha}{\sqrt{\sum_{j=1}^{D} |f_i|^{2\alpha}}}
\]  

(7.1)

The motivation behind this is to change the distribution of feature vectors in features space. For \( \alpha < 1 \), feature vectors which only have few components with high magnitude (\( |f_{a\in L}| \gg |f_{j\notin L}| \) for \( |L| \ll D \)) get regularized in the sense that the high magnitude components become smaller and low magnitude components become bigger. For \( \alpha > 1 \) the opposite happens. In Figure 7.2 we show the influence of \( \alpha \) for different datasets and region detectors on the final evaluation metric (mAP/top-4). One can clearly see that a significant improvement over \( \alpha = 1 \) is achieved, and further yet \( \alpha \approx \frac{1}{2} \) yields the biggest improvement for all datasets and all tested region detectors. This result is very nice, because \( \alpha = \frac{1}{2} \) scaling seems quite universal and is very easy to implement into any local feature based image retrieval system. Additionally these
result also fall well in line with the results of other researches like [Arand-jelović and Zisserman 2012].

Figure 7.2: Evaluation results with different feature exponents for affine adapted interest points and SIFT descriptors.

7.3 Document similarity function

The key component for ranking images in a bag-of-words framework are document similarity functions $\text{sim}(\vec{q}, \vec{d})$. For a query vector $\vec{q}$ and a database vector $\vec{d}$, the similarity function can be defined as: $\text{sim}(\vec{q}, \vec{d})$.
the similarity function $\text{sim}(\vec{q}, \vec{d})$ should be high if $\vec{q}$ and $\vec{d}$ correspond to similar images and low if they correspond to dissimilar images. There are several popular choices for similarity functions and most of them follow the term frequency-inverse document frequency (tf-idf) scheme which was originally introduced in the context of text retrieval (cf. for example [Baeza-Yates and Ribeiro-Neto 1999]). As the choice of similarity function may impact retrieval performance, we take a look at several combinations of popular similarity functions and inverse document frequency terms.

Let $\vec{q}$ and $\vec{d}$ be $V$-dimensional query and database vectors ($V$ being the size of the visual vocabulary), where $q_i$ or $d_i$ count the occurrences of visual word $i$ in the query- or database-document respectively. We define the following three similarity functions:

- **COS** is the cosine similarity, which calculates the cosine of the angle between to documents:

$$\text{sim}(\vec{q}, \vec{d}) = \frac{\sum_{i=0}^{V} q_i d_i \text{idf}(i)^2}{\sqrt{\sum_{i=0}^{V} (q_i \text{idf}(i))^2} \sqrt{\sum_{i=0}^{V} (d_i \text{idf}(i))^2}}$$  \hspace{1cm} (7.2)

- **COS2** is a cosine similarity variant, where the $\text{idf}$ terms are omitted in the denominator:

$$\text{sim}(\vec{q}, \vec{d}) = \frac{\sum_{i=0}^{V} q_i d_i \text{idf}(i)^2}{\sqrt{\sum_{i=0}^{V} q_i^2} \sqrt{\sum_{i=0}^{V} d_i^2}}$$  \hspace{1cm} (7.3)

This similarity function was suggested by [Jegou et al. 2008].

- **IOU** is the intersection over union similarity measure:

$$\text{sim}(\vec{q}, \vec{d}) = \frac{\sum_{i=0}^{V} \min(q_i, d_i) \text{idf}(i)}{\sum_{i=0}^{V} \max(q_i, d_i) \text{idf}(i)}$$  \hspace{1cm} (7.4)

This similarity measure has some importance, as it induces the min-hash locality sensitive hashing scheme which is popular for large scale document clustering [Chum and J.Matas 2008].

The $\text{idf}(i)$ functions which appear in the similarity functions add a weight to each visual word. This is motivated by the idea, that rare visual words hold
more information than frequently occurring visual words, and therefore \( \text{idf}(i) \) should give a larger weight the rare visual words. Commonly \( \text{idf}(i) \) functions are defined as the negative logarithm of some measure of how often a visual word occurs in different documents, therefore the name inverse document frequency-term. We evaluated several \( \text{idf}(i) \) variants which appear in the image retrieval literature. Let \( d \in D \) be the set of all database vectors, then we define the following four \( \text{idf}(i) \) functions:

- **IDF1** was used by [Nistér and Stewénius 2006]

  \[
  \text{idf}(i) = \log \left( \frac{|D|}{|\{d \in D | d_i > 0\}|} \right) \tag{7.5}
  \]

- **IDF2** was used by [Sivic and Zisserman 2003]

  \[
  \text{idf}(i) = \log \left( \frac{|D|}{\sum_{d \in D} d_i} \right) \tag{7.6}
  \]

- **IDF3** was used by [Jegou et al. 2008].

  \[
  \text{idf}(i) = \log \left( \frac{\sum_{d \in D} \sum_{j=0}^{V} d_j}{\sum_{d \in D} d_i} \right) \tag{7.7}
  \]

Additionally, as we have seen, that an exponential scaling of the feature vectors significantly improves results, we attempt, like in [Perronnin et al. 2010], to transfer this idea to document similarity functions by adding a \( \beta \)-exponential scaling of the \( \text{idf}(i) \) weighted bag-of-word vectors:

- **COS**

  \[
  \text{sim}(\vec{q}, \vec{d}, \beta) = \frac{\sum_{i=0}^{V} q_i^\beta d_i^\beta \text{idf}(i)^{2\beta}}{\sqrt{\sum_{i=0}^{V} (q_i^\beta \text{idf}(i))^{2\beta}} \sqrt{\sum_{i=0}^{V} (d_i^\beta \text{idf}(i))^{2\beta}}} \tag{7.8}
  \]

- **COS2**

  \[
  \text{sim}(\vec{q}, \vec{d}, \beta) = \frac{\sum_{i=0}^{V} q_i^\beta d_i^\beta \text{idf}(i)^{2\beta}}{\sqrt{\sum_{i=0}^{V} q_i^{2\beta}} \sqrt{\sum_{i=0}^{V} d_i^{2\beta}}} \tag{7.9}
  \]
7. Engineering Aspects

Figure 7.3: Evaluation results with different scoring and \( idf \) functions

- **IOU**

\[
\text{sim}(\vec{q}, \vec{d}, \beta) = \frac{\sum_{i=0}^{V}(\min(q_i, d_i)idf(i))^\beta}{\sum_{i=0}^{V}(\max(q_i, d_i)idf(i))^\beta}
\]  

(7.10)

In cases where \( \beta > 1 \), higher weight given to document vector components with high magnitude, while for \( \beta < 1 \) it is decreased. Figure 7.3 shows the resulting mAP/top-4 values for all possible combinations of the described \( \text{sim}(\vec{q}, \vec{d}) \) and \( idf(i) \) functions and different values of \( \beta \). The first observation we make, is that there is only little difference between \( IDF1/IDF2 \) and \( IDF3 \). Therefore it does not seem to matter whether the denominator counts the number of documents that contain visual word \( i \) at least once or how often visual word \( i \) appears in the database. However as \( IDF1 \) and \( IDF2 \) seem clearly su-
7.3. Document similarity function

Figure 7.4: (a) Illustration of the voting scheme inside one voronoi cell. Left side: Multiple features inside the same cell lead to an overcounting. Right side: Ideal case, without overcounting. (b) Scaling term \( \frac{v}{(1+v)^2} \)

Prior to IDF3, it does make a big difference if either denominator is chosen to be the number of documents or the number of visual words.

The second observation is that while the IOU similarity measure is often better than the COS and COS2 for \( \beta = 1 \), COS and COS2 clearly outperform IOU for \( \beta < 1 \). Furthermore we again see that on all datasets except for the Holidays dataset performance is best for \( \beta \approx \frac{1}{2} \). We believe the explanation for this, is that when we consider the similarity functions as voting schemes, where votes are cast for each component of the query and document vectors, then taking the square root of the components implicitly linearizes the voting scheme. We can illustrate this using an example. Consider a situation where \( q_i = d_i = 2 \). This means that both the query image and the database image have two feature vectors which fall into the same voronoi cell of the visual vocabulary as illustrated in figure 7.4a. Taking the product of \( q_i d_i \) corresponds to both query features casting votes to both document features, which give 4 votes in total and intuitively seems like an over counting. Taking the square root \( \sqrt{q_i d_i} = 2 \) in this example gives a more sensible result. Formally we can also show, that indeed a linearization takes place. Let without loss of generality \( q_i \leq d_i \) and let’s define total amount of visual features that fall into the same voronoi cell \( s \) and the ratio of the number of query features and document features \( v \):

\[
\begin{align*}
s &= q_i + d_i \\
v &= \frac{q_i}{d_i} \in [0 \ldots 1]
\end{align*}
\]
We can rewrite the product $q_i d_i$ as

$$q_i d_i = s^2 \frac{v}{(1 + v)^2}$$  \hspace{1cm} (7.13)

which makes it clear that $q_i d_i$ is a quadratic function in the total number of visual words in the Voronoi cell which an additional scaling term $\frac{v}{(1+v)^2}$, which accounts for uneven distributions in the number of features of the query and document image. A plot of the scaling term is shown in Figure 7.4b. For $v = 0$ where one document has no visual words, the scaling term is zero, therefore the features in the Voronoi cell do not contribute to the overall similarity. The function then monotonically increases to a maximum at $v = 1$ where both, the query and document image have equal number of visual words. Therefore taking the square root linearizes the voting scheme in terms of $s$:

$$\sqrt{q_i d_i} = s \frac{\sqrt{v}}{(1 + v)}$$  \hspace{1cm} (7.14)

### 7.4 Conclusion

We have demonstrated, that many publicly available binaries for feature extraction do not work well “out-of-the-box”, so we advise any researchers new to the field of image retrieval to first carefully vet feature extraction binaries before using them in their experiments. Further more we evaluated several popular local feature detectors and descriptors in great detail using the Oxford5k, Paris, Kentucky and Holidays datasets. With which we demonstrated, that the choice of the local feature detector and its subsequent descriptor have a significant effect on retrieval performance. Additionally, we show that a simple modifications to the the original feature vectors as well as the choice of the ranking function significantly improves retrieval accuracy.
Conclusions and Outlook

In this thesis we have presented several exciting new applications that arise from the combination of large scale image retrieval and large scale image mining and demonstrated them with fully functional prototype implementations. And we have looked at the integration of visual search on mobile phones using a see through augmented reality user interface. Further contributions include methods to improve large scale image retrieval with little computational overhead as well as a detailed evaluation of local feature detectors, modified local features as well as different ranking functions in the bag-of-words framework.

8.1 Contributions

In Chapter 3 we presented a fully automatic annotation pipeline for holiday snapshots. The system automatically collects data from community photo collections and does object-level annotation with bounding boxes as well as geo- and text-tag extraction and also finds relevant Wikipedia articles. The mining method produces groups photos of objects into object clusters which serve as exemplar models for items in the database the object clusters allow us not only to recognize query images for annotation from a wide variety of viewpoints, but also to automatically localize the object in database images. Furthermore we have shown, that this information can be used to reduce the number of visual words that have to be inserted into the image index without any loss in retrieval precision. Ultimately the design of the system allows for scaling up to several millions of images in the reference database. And finally the system was evaluated on challenging test-data and a large database in terms of correct recognition and annotation of objects as well as their localization within the query image.
Chapter 4 addresses shortcomings in the image mining part of Chapter 3. The entire method is again fully automatic and requires no user interaction. By using cross-media information, we automatically generated text strings that serve as queries for Internet search engines in order to find additional images of a given object. This allows for expansion of small image clusters with additional images. Beyond this we show how redundant information in large image clusters can be removed with a simple graph based approach. The combination of expanding small image clusters and reducing large image clusters results in better recognition performance and lower computational and storage cost. We have also shown that it is possible to exploit the wisdom of crowds to a-priori determine if a potential text query may be useful for retrieving additional images. Finally, while this Chapter focused on object recognition, we demonstrated that the cluster expansion method is potentially valuable for unsupervised 3D reconstruction.

Chapter 5 building on the previous two Chapters demonstrated a fully functional augmented reality system which recognizes and tracks stationary as well as mobile objects. It combines a client-side tracker on a mobile device and with a server-side object recognition service introduced in Chapters 3 and 4. The client-side tracker allows for real-time tracking and interactive usage on state of the art smart phones. And our server-side implementation of the recognition service provides an object recognition service for 300000 object clusters in augmented reality compatible response times strictly under 2 seconds. It uses a memory efficient geometric verification step which has almost equal precision to geometric verification done on raw local feature vectors. In our whole set-up we do not use any markers and do not require GPS information for object recognition and tracking. The capabilities of the system are demonstrated with a prototype application on the Android platform, which is able to augment both stationary (landmarks) and non stationary (media covers) objects.

In Chapter 6 we taken a step away from practical applications, but focused more on improving image retrieval in general. We demonstrated that using information provided by \( k \)-reciprocal nearest neighbors that a significant improvement in bag-of-words retrieval can be achieved, without considering the geometric arrangement of features in an image nor by modifying the feature quantization step. Our method uses \( k \)-reciprocal nearest neighbors to identify an initial set of highly relevant images in the database which are then used to re-rank the remaining part of the database. On many data sets our approach competes with the state of the art. The memory overhead of our method is
linear in the number of documents while the average query time overhead is neglectable.

In Chapter 7 we have demonstrated, that the choice of the local feature detector and its subsequent descriptor can have a significant effect on retrieval performance. We evaluated several popular local feature detectors and descriptors in great detail using the Oxford5k, Paris, Kentucky and Holidays datasets. Additionally, we show that several modifications of the original feature descriptor as well as the choice of the ranking function significantly improves retrieval accuracy.

### 8.2 Outlook and open questions

Landmark or in general Object recognition in a multimodal context has become a very exiting area of research, as there is a massive amount of data freely available on the Internet. We have shown mining techniques may be just as important as the underlying image retrieval techniques. And like others before we believe in the motto more data simply gives you better results. While this statement is quite generally well accepted and may seem somewhat trivial it leads to exciting questions. For instance as we nowadays have a truly overwhelming amount of data at our fingertips with the Internet, how can one handle billions if not trillions of images? In Chapter 4 we have shown that simple approaches can already greatly reduce the amount of data without any loss in retrieval performance, but there must be a more efficient way to do so.

The reciprocal nearest neighbours approach presented in Chapter 6 works well for datasets where multiple images of the same object are present. Given that a dataset might not have this characteristic, how could one still make use of reciprocal nearest neighbours? One possibility of course would be to artificially generate new images by transforming the original images. Or if metadata is available for the dataset, then possibly an automatic mining procedure like in Chapter 5 may help. All of this gives raise to potential future work. More generally, the reciprocal nearest neighbours approach looks at the structure of the document space, which ultimately yielded significant improvements for image retrieval. Therefore it seems that much can be gained from better understanding the document space structure, as it is highly anisotropic. As the reciprocal nearest neighbour approach is purely datadriven, it becomes more expensive the more images are indexed. Might there be a way to describe the document
space structure more efficiently? Assuming there is, then large amounts of
data, be it mined from the Internet or artificially generated, might be used to
learn this structure possibly yielding better results at lower computational and
storage overhead.

Ultimately one must ask, is the bag-of-words approach really the one with the
brightest future? Looking at the timeline of how bag-of-words was improved
over the years, it becomes clear, that one key issue is the proper representation
of quantized feature vectors. Using $k$-means with a very large $k$ gives a low
quantization error, but due to the many degrees of freedom often leads to an
overfitting. This is why it makes a big difference if the vocabulary is trained
on the test set or an independent set. Techniques such as Hamming Embed-
ding [Jegou et al. 2008] already alleviate such problems somewhat, but come
at the cost of memory overhead, which is the biggest issue for scalability. Per-
haps there is a more robust, but still memory efficient quantization scheme for
local features.

Further more one should note, that for instance [Jégu et al. 2011b] have
already taken a step away from bag-of-words and demonstrated image re-
trieval on 10 million images on a single laptop. While their approach per-
forms best for images which have global similarity unlike for instance the
images from the Oxford5k dataset, the results are still very impressive. One
reason the bag-of-words approach works well, is that it’s inherently based
on local image attributes, which are much easier to describe in a viewpoint
invariant manner as opposed to larger patches which have a more compli-
cated transformation behaviour. However there should be a sweet spot be-
tween global image description and local image description and I hope that
future researchers in computer vision set out to find it, return with impressive
results and are greeted with applause from the computer vision community.
Bibliography


[Bay et al. 2006] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *ECCV’06*, 2006. 2, 2.1, 3.4, 1, 4.2.2, 4.3.1, 5.1, 5.3.1, 6.3


[Bosch et al. 2006] A. Bosch, A. Zisserman, and X. Munoz. Scene classification via pLSA. In ECCV’06, 2006. 3.2


[D.Wagner and Schmalstieg 2003] D. Wagner and D. Schmalstieg. First steps towards handheld augmented reality. In ISWC’03, 2003. 5.1


[goo] lhttp://www.google.com/mobile/goggles/. 5.1


[inr] http://lear.inrialpes.fr/~jegou/data.php. 6.5.1


[Klein and Murray 2008] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In ECCV’08, 2008. 5.1


[Lowe 2004] D. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 60(2), 2004. 2.1, 2.4, 5.1, 5.3.1, 7.1.1


[nis] http://www.vis.uky.edu/~stewe/ukbench/. 6.5.1


[oxf] http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/index.html. 6.3, 6.5.1

[par] http://www.robots.ox.ac.uk/~vgg/data/parisbuildings/index.html. 6.5.1


[Quack et al. 2008] T. Quack, B. Leibe, and L. Van Gool. World-scale mining of objects and events from community photo collections. In *CIVR ’08*, 2008. 3.1, 3.2, 3.3, 3.2, 3.3, 4.1, 4.2.1, 4.2.1, 4.2.1, 4.3.1, 4.3.2


[Schaffalitzky and Zisserman 2002] F. Schaffalitzky and A. Zisserman. Multi-view matching for unordered image sets, or ”how do i organize my holiday snaps?” In ECCV’02, 2002. 3.2


[Simon et al. 2007] I. Simon, N. Snavely, and S. M. Seitz. Scene summarization for online image collections. In ICCV’07, 2007. 3.2, 4.1, 4.2.1


5.1


[Wagner *et al.* 2008a] D. Wagner, T. Langlotz, and D. Schmalstieg. Robust and unobtrusive marker tracking on mobile phones. In *ISMAR’08*, 2008. 5.1, 5.4.2


[wik] http://www.wikitude.org. 5.1, 1


Curriculum Vitae

Personal data

Name: Stephan Gammeter
Date of Birth: April 18\textsuperscript{th} 1983
Nationality: Swiss and American

Education

11/2007–05/2012 PhD in Computer Vision
at the Computer Vision Laborartory of ETH Zurich.

05/2007–09/2007 Diploma Thesis in Neuroscience
”Spike-Sorting with Hidden Markov Models”.
at the Institute of Neuroinformatics University
of Zurich and ETH Zurich.

Specialization in Computational Physics
and Neuroscience.

08/1995–01/2002 Realgymnasium Rämibühl

Work experience

5/2012–present Google Switzerland GmbH
Software engineer.

12/2006–5/2012 Xylix Software GmbH
Co-founder of Xylix Software, a company developing
Information Retrieval Systems.

Developing software for military radio network planning.