Diss. ETH No. 18190

Large-Scale Mining and Retrieval of Visual Data in a Multimodal Context

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Technical Sciences

presented by

Till Quack
MSc. ETH Zuerich
born 15. September 1978
citizen of Germany

accepted on the recommendation of

Prof. Dr. Luc Van Gool, examiner
Prof. Dr. Andrew Zisserman, co-examiner

September 2008

To my grandparents.

Abstract

In recent years significant progress has been made in the field of object recognition, mostly due to the introduction of powerful local image features. At the same time, a growing amount of images and videos are being shared on the Internet. This dissertation tries to combine these developments in proposing efficient retrieval and mining algorithms suitable for such visual data, while exploiting its multimodal context.

The work at hand advances the state-of-the-art with three main contributions. Firstly, with the investigation of itemset mining algorithms in the domain of visual data. This class of simple, but efficient algorithms have proven to be a useful tool for other kinds of data. We adapt these methods to work with local visual features. The resulting algorithms are successfully employed to mine specific objects in video data, and to identify frequent feature configurations as representatives of object classes.

The second contribution consists of a multimodal data-mining method, which automatically mines objects and events from community photo collections on the Internet. After crawling geotagged photos, the method automatically clusters photos showing the same object or event using visual features. The system then proceeds with analyzing the multimodal context of each identified cluster, in particular text associated with the individual photos. This analysis results in a textual description of the clusters. Furthermore, it is used to identify related Wikipedia pages. Finally, building again on the mined visual data, this assignment is verified, and refined up to an object-level annotation of mined entities for applications such as retrieval or auto-annotation.

The third and final contribution consist of several prototype applications for scalable retrieval in visual data, partly building on the data mined in the previous steps. These retrieval applications focus on applications for mobile devices, again including multimodal context such as GPS location of the user. In addition to the mobile retrieval applications, novel web-and desktop applications are designed, for browsing and auto-annotation in personal photo collections.

Zusammenfassung

In den letzten Jahren wurden erhebliche Fortschritte im Bereich der Objekterkennung erzielt. Diese Fortschritte basierten zu einem grossen Teil auf der Einführung sogenannter lokaler Bildmerkmal Detektoren und Deskriptoren. Im gleichen Zeitraum wurden rasant wachsende Mengen von digitalen Bildern auf dem Internet zugänglich gemacht. Die vorliegende Arbeit hat zum Ziel diese Entwicklungen zu kombinieren, indem sie effiziente Such- und Mining Algorithmen unter einbeziehung des multimodalen Kontextes analysiert.

Damit werden folgende Beiträge zum aktuellen Stand der Forschung geleistet. Ein erster Beitrag besteht aus der Untersuchung der Anwendbarkeit vom itemset mining Algorithmen im Bereich der visuellen Daten. Diese Klasse von einfachen, aber effektiven Algorithmen wurde bereits in anderen Gebieten erfolgreich angewendet. Wir passen die Methoden an das Problem des Minings in Bilddaten an und zeigen ihre erfolgreiche Anwendung um Objekte in Videos zu minen und um signifikante Feature Konfigurationen als Repräsentanten für Objektklassen zu ermitteln.

Ein zweiter Beitrag besteht aus der Einführung einer multimodalen mining Methode, welche vollautomatisch Objekte und Ereignisse aus Community Photo-Plattformen aus dem Internet detektiert. Nach einem crawling Prozess basierend auf geo-referenzierten Bildern, ermittelt die Methode Cluster von Bildern, welche das gleiche Objekt abbilden. Im folgenden Schritt analysiert das System den multimodalen Kontext jedes Clusters, insbesondere Textfragmente, die mit den Bildern im Cluster in Verbindung stehen. Diese Analyse resultiert in einer Beschreibung des Clusters mittels Worten. Die Methode findet ausserdem automatisch relevante Artikel aus Online Enzyklopdien für die Cluster. Basierend auf diesen Daten wird ein System für Auto-annotation von Photos auf dem Objektlevel eingeführt.

Der dritte und letzte Beitrag besteht aus mehreren Prototypen für Bildsuche unter besonderer Berücksichtigung mobiler Endgeräte. Hier wird wieder der multimodale Kontext berücksichtigt, beispielsweise mittels Einbezu der GPS Ortung des Benutzers.

Acknowledgements

I am grateful to a number of wonderful people who supported me during the time this dissertation came into existence.

First and foremost I thank my advisor Prof. Dr. Luc Van Gool, for offering great scientific freedom, and always being available when his invaluable expertise, guidance and advice were required. Particular thanks go to Prof. Dr. Andrew Zisserman for being aware of my work and agreeing to co-referee this thesis.

I received extraordinary support from Prof. Dr. Vittorio Ferrari and from Prof. Dr. Bastian Leibe. Both were irreplaceable through their availability for countless fruitful discussions, hands-on support in late nights before paper deadlines, and being a constant source of motivation.

My colleagues at the Computer Vision Lab at ETH Zurich provided a joyful and energetic atmosphere. Particular thanks go to Andreas Ess, Stephan Gammeter, Raphael Hoever, Tobias Jaeggli, Alain Lehmann, Stefan Saur, and Thibaut Weise, for being great pals.

I also thank my Semester and Diploma students, who made valuable contributions to parts of this work.

I am particularly thankful to my friends at our startup kooaba. Being able to apply and extend some of the research conducted in this thesis in a business is a unique experience. This wouldn't be possible without a fantastic group of people, especially Dr. Herbert Bay, who had the courage to start this adventure with me in the first place.

I thank the Sander family for providing not only a quiet and wonderful place to write a large part of this thesis, but also making me feel at home.

I am eternally grateful to my parents Roswitha and Martin, for 30 years of love and support, and my brothers Niels and Manfred, who are still my best friends. Finally, my very special and heartfelt thanks go to Andra, without whom simply nothing would be the same.

Contents

List of Figures

List of Tables

1

Introduction

"Where is the wisdom we have lost in knowledge? Where is the knowledge we have lost in information?" T. S. Eliot, The Rock (1934)

Understanding the contents of an image is one of the fundamental problems of Computer Vision research. It is also a topic of increasing importance. While at the beginning of this thesis in the year 2004 around 50 million digital cameras were sold, the number is expected to surpass 100 million devices this year. Hundreds of millions of cameras in use produce a large amount of digital photos. People share these photos on digital platforms on the Internet, allowing millions of visitors to access their photo collections.

Computer Vision methods can simplify access to the data in these large visual repositories. Especially methods which allow for identification of objects in images are useful tools for easing organization and search. The analysis of images at the object-level is typically divided into two subtasks: detection of specific objects and object class detection. A specific object could for instance be a landmark building such as the Eiffel tower, or a specific car. Examples of occurrences of object classes are the presence of *a* building or *a* car in an image, no matter which building or which kind of car.

Research in both fields has made significant progress in recent years. This thesis builds on this research and tries to extend it towards applications in the context of the Internet. This endeavor comes with several open questions. The first challenge is posed by the large amounts of data. Methods which can be applied to larger amounts of data have to be efficient and scalable. We thus attempt to employ methods from data mining for the task of object detection. These methods are known to scale in other fields, but are somewhat simpler than methods commonly used for object detection. This raises the question, if they can compete with the state-of-the-art in object detection.

Another unique characteristic of the visual data shared on the Internet is that it is often embedded in a multimodal context. A photo has been taken by a certain

user, at a certain location and time. This is particularly relevant for photos taken with mobile phone cameras. Furthermore, images are often embedded in text or sometimes labeled with keywords. Can we exploit this multimodal redundancy of descriptions to learn objects and their descriptions from the data? This is the second question we investigate in this thesis.

Finally, apart from all scientifically motivated interest in the recognition of objects in images, we also want to let real-world applications drive our research. Relating to the introductory quote of this dissertation, the constant flow of digital information on the Internet poses challenges to us as individuals. Enormous amounts of information are only a mouse click away, but the efficient extraction of desired or relevant information is an increasingly difficult task. Thus, we try to create prototypes of applications, which help us to analyze and search large repositories of visual data. With the hope, that they might eventually lead to applications, which assist us in gaining knowledge from the constant information flow we are faced with every day.

1.1 State of the Art in Object Recognition

While the topics dealt with in this thesis cover a variety of fields, the common denominator is always the goal of recognizing objects in images. In that sense, this dissertation builds on over 40 years of research. What follows is a brief summary of the field, with the intent to give the reader a general orientation in this rather wide area of research. Throughout the thesis, we will discuss more specific related work at the end of each chapter.

Object recognition methods can be roughly classified into geometry based and appearance based method. Geometry based approaches try to model the (3D) characteristics of objects using global object properties, most commonly the object's contours. The task of detecting the an object then corresponds to identifying the model and its pose, which might have generated the features observed in the image. Notable works using this paradigm include [Grimson and Lozano-Pérez, 1987; Wolfson and Rigoutsos, 1997; Lowe, 1991]. The main disadvantages of model-based methods include the requirement of a possibly rather complex 3D model and the difficulty in detecting and interpreting its (contour-based) features. These are probably some of the reasons why recent research has focussed mostly on appearance based methods.

Appearance based methods are further subdivided into global appearance methods and local appearance methods. Both have in common that they don't rely on a 3D model for recognition, but base recognition on (sample) images of the object only. Global methods attempt to derive a compact representation of the objects appearance from an entire image, *e.g.* by so-called eigenimages (dimensionality reduced

representations of the images in a manifold) [Murase and Nayar, 1995], by matching templates [Dufour *et al.*, 2002], or color histograms [Swain and Ballard, 1991].

The disadvantages of the global appearance based methods are, that they are not robust to clutter in a scene and partial occlusions of the object. Local appearance methods try to overcome these challenges by treating an object as a collection of localized parts, or local features. Each local feature is expressed as a descriptor vector of the appearance of the corresponding image part. (A more detailed description of common local feature types is given in Chapter 2.1). A database of model images is then represented as a collection of these vectors. The presence of a database object in a query images is determined by first extracting local features from the query image, and then searching for the nearest neighbor of each local feature patch in the database. The collection of matched features will cast votes for a particular object in the database. By relying on local image patches this approach is robust toward occlusion and clutter. Relying on geometric constraints for the possible location of features in the image plane can further improve recognition. Examples of early works building on this paradigm are [Schmid and Mohr, 1997; Lowe, 1999]. Later on, local appearance based features were not only used to recognize specific objects, but also to recognize the presence and even localize instances of object classes in images [Weber *et al.*, 2000a; Agarwal and Roth, 2002]

Local appearance based methods are at the heart of most state of the art object recognition methods and are also the basis for the research carried out in this thesis. Thus, a more detailed discussion of the more recent work in this field is given in Chapter 2.3.

1.2 Contributions of this thesis

The work at hand advances the state-of-the-art in object-level retrieval and mining with three main contributions. Firstly, with the investigation of itemset mining algorithms in the domain of visual data. This class of simple, but efficient algorithms have proven to be a useful tool for other kinds of data. We adapt these methods to work with local visual features. The resulting algorithms are successfully employed to mine specific objects in video data, and to identify frequent feature configurations as representatives of object classes.

The second contribution consists of a multimodal data-mining method, which automatically mines objects and events from community photo collections on the Internet. After crawling geotagged photos, the method automatically clusters photos showing the same object or event using visual features. The system then proceeds with analyzing the multimodal context of each identified cluster, in particular text associated with the individual photos. This analysis results in a textual description

of the clusters. Furthermore, it is used to identify related Wikipedia pages. Finally, building again on the mined visual data, this assignment is verified, and refined up to an object-level annotation of mined entities for applications such as retrieval or auto-annotation.

The third and final contribution consist of several prototype applications for scalable retrieval in visual data, partly building on the data mined in the previous steps. These retrieval applications focus on applications for mobile devices, again including multimodal context such as GPS location of the user. In addition to the mobile retrieval applications, novel web-and desktop applications are designed, for browsing and auto-annotation in personal photo collections.

1.3 Organization of this thesis

This thesis is organized as follows:

Chapter 2 introduces various basic methods and algorithms which are important throughout this work. Most importantly they include local visual feature types, and a more detailed discussion of local appearance based methods for object (-class) detection, with a special emphasis on visual vocabulary based approaches.

Chapter 3 describes our work with itemset mining algorithms in the domain of visual data. The goal here is to come up with a method for efficient detection of re-appearing structures of local features, using data-driven mining algorithms rather than explicit model learning. The resulting methods are applied to and evaluated on tasks in video mining and object class detection using standard benchmark data.

Chapter 4 takes mining from the feature level to the object level. We introduce a method to mine objects and events from community photo collections on the Internet. The approach relies on geotagged photos, which are clustered based on their similarities calculated from local feature matches. Beyond the visual cues we extend our mining method to include cues from other modalities such as the textual tags describing the photos. This allows for labeling of the mined objects and events. Furthermore, using multimodal information from Wikipedia, we relate Wikipedia articles to the identified object clusters using a multimodal matching and verification procedure. Finally, we demonstrate how the mined data can be used to derive object-level auto-annotations of objects such as landmark buildings in holiday snaps. Experiments are conducted on hundreds of thousand of photos downloaded from the Internet.

Chapter 5 deals with the user- or application-centric aspect of object recognition. We demonstrate several prototypes for object recognition applications, with a special focus on mobile devices. Several options for user interaction with the system are investigated. The mobile applications are complemented with two applications for the desktop or the web, namely auto-annotation and 3D reconstruction – both applications build directly on the results from Chapter 4. Finally, we introduce a method to localize and read text in natural images, with the goal, to make this information also accessible to visual retrieval systems.

Chapter 6 discusses methods which allow to scale object-level retrieval to large amounts of data in the order of up to 1 million images. We investigate, which properties make nearest neighbor search for databases of local features different from "general purpose" nearest neighbor search. We then evaluate three methods (LSH, Redundant Bit Vectors, and Metric Trees) under that aspect. Metric trees are further extended to form forests of metric trees and their performance is compared to state of the art visual vocabulary based methods.

Chapter 7 concludes by discussing the results of our work and pointing out further research directions based on our findings.

Since our work touches several independent research areas, each of the main chapters $4-6$ contains a section discussing related work specific for the topic of that chapter.

2

A Set of Tools

In this chapter we introduce some basic tools and algorithms we build on throughout our work. They include local feature extractors and descriptors, clustering algorithms, visual vocabularies, itemset and mining methods, and classifier boosting.

2.1 Local Feature Detectors and Descriptors

The introduction of very powerful local visual features in the late 90's is probably one of the main reasons for the astonishing progress the field of computer vision has made in recent years. Unlike global features (*e.g.* variations of global color histograms or texture features), which describe the entirety of an image with a single feature vector, local features decompose the image into localized image patch descriptors around interest points. An example is shown in Figure 2.1. Selecting the "right" image patches, and describing them in a "meaningful" way is the important contribution of the research that led to the local features we can build on in our work. Describing an image with local features typically consists of two steps: interest point detection, and construction of a descriptor for the image patch around the interest point.

A good interest point detector locates points, that can be detected repeatedly, even if the original image is modified or the same scene is depicted under varying conditions. Such variations include *e.g.* viewpoint changes (angle, zoom, *etc.*), lighting changes, or image compression. The main criterion to judge the quality of an interest point detector is thus its invariance to those perturbations, which is typically measured with a repeatability value [Mikolajczyk and Schmid, 2004a], expressing if the same interest point can be reliably detected at the same position, even after an image has undergone transformations. While locating "interesting" points in images has a long history in computer vision (*e.g.* with Harris corners [Harris and Stephens, 1988]), achieving discriminance, reliable localization, and robustness to scale or affine

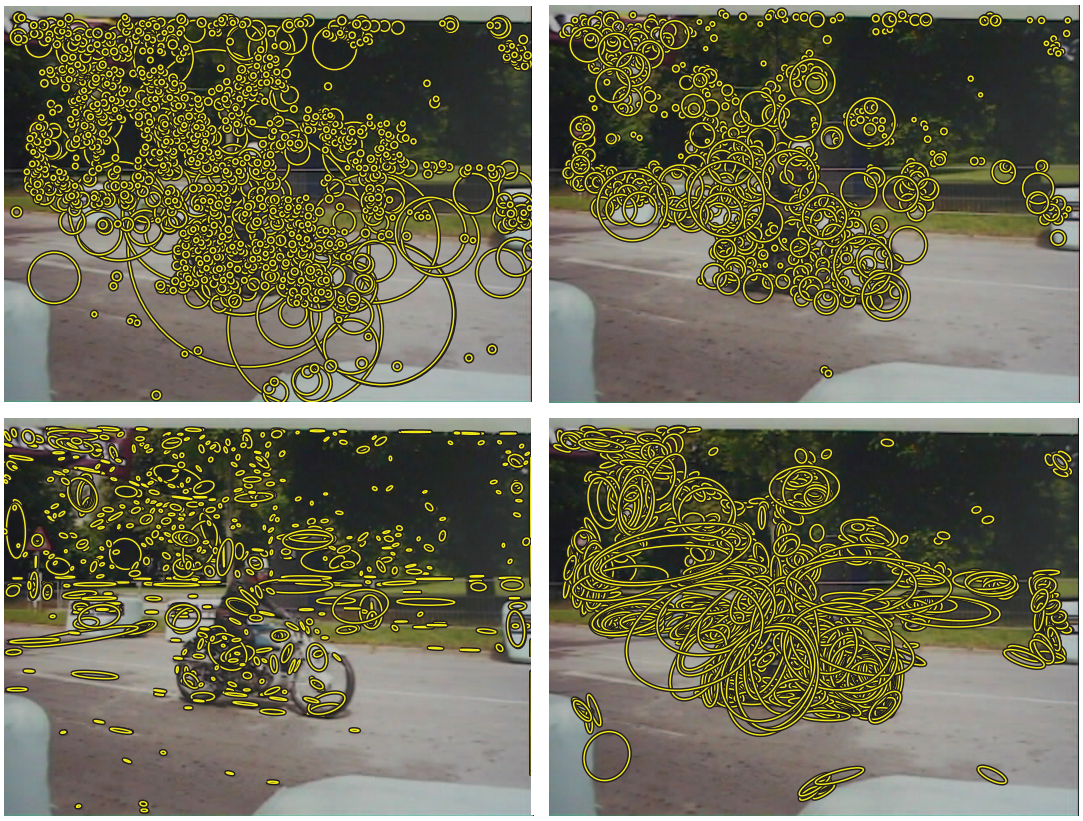


***Figure 2.1**: Examples of local features. SIFT, SURF, MSER, and Hessian-Affine (clockwise from top-left).*

changes is quite challenging. A good overview and comparison of some of the most well-known interest point detectors can be found in [Mikolajczyk and Schmid, 2004a; Mikolajczyk *et al.*, 2005; Tuytelaars and Mikolajczyk, 2008].

After localizing an interest point, a region around it is usually encoded using a descriptor vector, *e.g.* based on the histogram of gradients observed in the image patch. The most important quality criteria for descriptors are a compact representation and high precision and recall when matching descriptors from a database of images (*i.e.* finding the right point correspondences and finding all point correspondences). An evaluation of some of the most well-known interest point descriptors can be found in [Mikolajczyk and Schmid, 2005].

Comparing two images using local features boils down to the execution of the following steps:

1. Feature extraction: extract interest points and their descriptors.
2. Feature matching: for each interest point find its corresponding features in the other image, or from a database of images. This involves often finding the

nearest neighbor(s) of a feature descriptor from a large database of reference images.

3. Recognition: based on the number and location of the matches decide, if the two images show the same object or scene. This step can include further verification using a model, either specific to an object(-class) or general verification models, *e.g.* multiple view geometry.

Steps 2 and 3 are known as the *correspondence problem*. While most of this thesis focuses on the third step, we also have a look at feature matching in Chapter 6, where we discuss options for scalable retrieval from large databases of local features.

The basic processing pipeline above has a variety of additional or modified steps depending on the application it is deployed for. Such applications include:

- 3D Reconstruction
- Image Mosaicking
- Object Recognition
- Object Class Recognition
- Image and Video Retrieval

Below we summarize the properties of some of the local feature types used in this thesis. They include features, which are invariant to scale changes (SIFT [Lowe, 2004] and SURF [Bay *et al.*, 2006b]) as well as features, which are invariant to affine changes (MSER [Matas *et al.*, 2002] and Hessian-Affine [Mikolajczyk and Schmid, 2004a]).

2.1.1 SIFT

SIFT (Scale Invariant Feature Transform) [Lowe, 2004] consists of both an interest point detector and descriptor. SIFT is scale and rotation invariant.

The interest point detector builds – as most other approaches for interest point detection – on scale-space theory, to obtain a scale-invariant interest point. This involves convolving the image with a Gaussian at several scales, creating a so called scale space pyramid of convolved images. Interest points are now detected by selecting points in the image, which are stable across scales. In the case of SIFT this is done using a Difference-of-Gaussians (DoG) approach, where the convolved images at subsequent scales are subtracted from each other. (The DoG approach is in fact simply an approximation of the Laplacian). Stable points are searched in these DoG

images by determining local maxima, which appear at the same pixel across scales. Afterwards, several refinement steps are applied, to select the most robust points (*e.g.* eliminating edge responses *etc.*). Finally, the most dominant orientations are determined, by creating a radial histogram of gradients in a circular neighborhood of the detected point. The maxima from this histogram determine the orientation of the point, and thus enable rotation invariance.

For the descriptor, around each interest point a region is defined, divided into orientation histograms on (4 x 4) pixel neighborhoods. The orientation histograms are relative to the keypoint orientation. Histograms contain 8 bins each, and each descriptor contains a 4x4 array of 16 histograms around the keypoint. This leads to a SIFT feature vector with (4 x 4 x 8 = 128 elements). This vector is normalized to enhance invariance to changes in illumination.

2.1.2 SURF

SURF [Bay *et al.*, 2006b] is a particularly fast and compact method. Just like SIFT, SURF is scale- and rotation invariant.

The interest point detector used by SURF is based on the Determinant-of-Hessian (DoH) blob detector. However, just as SIFT uses DoG as an approximation of the Laplacian, SURF uses a more efficient approximation of the Hessian. This is done using a courageous approximation of the Gaussian second order derivatives of the Hessian detector with simple box filters. Using box filters allows using integral images [Viola and Jones, 2001b] for efficient computation.

Just like its detector, the SURF descriptor is tuned for efficiency. It calculates a set of simple Haar-like features in sub-regions of a rectangular neighborhood around an interest point. As in the case of SIFT, this is done after determining a dominant orientation and expressing the descriptor in relation to that orientation to achieve rotation invariance. The Haar-like feature responses can again be calculated very efficiently using integral images.

2.1.3 Hessian-Affine

Hessian Affine interest point detectors [Mikolajczyk and Schmid, 2004a] belong to a class of so-called affine-covariant detectors, which are not only invariant to scale and rotation, but can even cope with affine changes. The main concept of these detectors is to find first a stable interest point in scale-space as with the methods described above, but afterwards to fit an elliptical region around the interest point. (Instead of a square or circle). This ellipse adapts – *i.e.* is covariant – with affine changes

of the underlying image structures. For Hessian-Affine detectors, the shape of this ellipse is determined with the second moment matrix of the intensity gradient.

Note, that the Hessian-Affine method is an interest point detector only, and does not come with it's own descriptor. It is thus typically combined with other descriptors, such as SIFT. The descriptor is extracted on a normalized region for all interest points, *e.g.* the ellipses are transformed into a circle, before the descriptor is calculated on the pixels within this circle.

2.1.4 MSER

MSER (Maximally Stable Extrema Regions) [Matas *et al.*, 2002] also belong to the class of affine-covariant detectors. They are not based on one of the 'standard' Gaussian scale space methods, but are based on connected components of an appropriately thresholded image. The word extremal refers to the property that all pixels inside the MSER have either higher (bright extremal regions) or lower (dark extremal regions) intensity than all the pixels on its outer boundary. The maximally stable in MSER describes the objective optimized during the threshold selection process: while changing the threshold value, these regions' binarization stays stable over a range of threshold values. "Maximally stable" is defined as the local minimum of the relative area change as a function of relative change of threshold.

Just as with the Hessian-Affine detectors, an ellipse can be fitted to the output regions of the detector, and after normalization, a region descriptor such as SIFT can be calculated on the pixels in the region [Mikolajczyk *et al.*, 2005].

2.2 Clustering

Data clustering involves partitioning a data set into groups of related items. The type of partitioning an algorithm is trying to achieve, and the strategy it uses to reach its goal depend strongly on the kind of data. Here, we consider clustering algorithms, which can operate with vector data. Their goal typically consists of identifying areas of high density (*i.e.* agglomerations of data points) in the space and forming clusters around them. Throughout our work we apply two of the most popular clustering methods, namely k-Means and hierarchical clustering, which are described in the following.

2.2.1 k-Means

One of the best known and widely used clustering algorithms is the k-Means algorithm [MacQueen, 1967]. It finds a partitioning of N points from a vector space

into $k < N$ groups, where k is typically specified by the user. The objective it tries to achieve is to minimize total intra-cluster variance, or, the squared error function

$$V = \sum_{i=1}^{k} \sum_{x_j \in c_i} (x_j - \mu_i)^2$$

where there are k clusters $c_i,\ i = 1 \dots k$, and μ_i is the mean of all the points $x_j \in c_i$.

The most common form of the algorithm uses an iterative refinement heuristic known as Lloyd's algorithm. The algorithm starts with an initialization of k centroids as representatives for the clusters. Based on a distance measure (the L_2 Norm is the correct distance to minimize the objective function, but other distances are often used, nevertheless), it assigns all points in the dataset to their closest centroid. Then it recalculates the centroid of each cluster as the mean of all data-points in the cluster. These steps are repeated until no points change clusters or a threshold (number of iterations, change of V) is reached. It's simplicity and rather fast execution times make k-Means a popular clustering algorithm.

While k is the only parameter that needs to be specified for k-Means, its choice is not trivial, in particular since it affects the outcome of the clustering result greatly. A common way to work around this problem is to just try several values for k. However, for large datasets this approach is too time-consuming because k can vary in a wide range (we might need a clustering with thousands of partitions) and the runtime of the algorithm for each k obviously increases with increasing number of datapoints N. (The time-complexity of the k-Means algorithm is $O(Nkld)$ for N datapoints of dimension d, and l iterations).

Another challenge is the initialization of the algorithm, which also has a substantial impact: for different initializations the algorithm may reach different results. k-Means is known to be vulnerable to getting stuck in local minima. It is not trivial to avoid this, so a common simple solution is to start the algorithm with different initializations and to keep the best outcome. Common variants of initialization include picking k random data-points, or stepwise selection of the farthest away data-point, beginning with the origin ("ping-pong" initialization).

Many improvements of the standard k-Means algorithm have been suggested [Farnstrom *et al.*, 2000; Elkan, 2003; Pelleg and Moore, 1999; 2000]. They either use efficient data structures or improve runtime and memory requirements by reducing the number of distance calculations based on some approximation criteria.

We tested the algorithm described in [Farnstrom *et al.*, 2000]. It is a single-pass algorithm with a buffer, *i.e.* it can work with limited memory and uses only one pass over the data-set. This is achieved by moving points, that wont change their cluster with high probability from memory to a retained set on the hard drive. However, it is an approximate method, which does not deliver the exact same results like standard

k-Means. (The method is related to the class of stream clustering algorithms, which build on the notion that data is received as a constant stream and can only be read once from a buffer).

Another method which significantly speeds up the exact k-Means algorithm (in particular for high values of k) is described in [Elkan, 2003]. Here, lower bounds of distances are determined based on the triangle inequality to avoid distance calculations. While the speed-up is impressive, it is traded for memory usage: The implementation requires keeping track of distance bounds in a table of dimensions $N \times k$.

In our work we use k-Means mostly to cluster local visual features into so-called visual vocabularies, see Section 2.3 of this chapter and Chapter 6.

2.2.2 Hierarchical Clustering

Unlike k-means, hierarchical clustering methods don't create clusters by iteratively climbing towards dense areas in feature space, but rather try to merge pairs of items and clusters successively, starting with the closest until some cutoff criterion is reached.

The clustering process begins by calculating all pairwise distances between data items $i, j \quad j \in N$. Then, starting with the smallest distance, pairs of items or clusters are merged. This way, a hierarchical cluster tree, or dendrogram, is created. On this dendrogram, clusters can be identified by "pruning" the tree at a certain level.

While merging pairs of clusters, the question arises, how the distance between two clusters A, B should be defined. In fact, there are several measures, the most popular being:

$$\begin{aligned} \text{single-link:} \quad & d_{AB} = \min_{i \in A, j \in B} d_{ij} \\ \text{complete-link:} \quad & d_{AB} = \max_{i \in A, j \in B} d_{ij} \\ \text{average-link:} \quad & d_{AB} = \frac{1}{n_i n_j} \sum_{i \in A, j \in B} d_{ij} \end{aligned}$$

Single-link merges clusters based on the distance of the two closest items in each cluster, complete link is based on the distance of the two items farthest away from each other, and finally, average-link takes the average distance as a criterion. Which distance measure is appropriate depends on the application.

The advantage of hierarchical clustering over k-Means is, that it can be applied to any kind of data, where a distance between two items can be defined. (Thus, also

any distance measure can be used). The main disadvantage is, that it relies on a calculation of pairwise distances between all items, which is $O(N^2)$. There are optimizations available, *e.g.* [Leibe *et al.*, 2008] rediscovered an optimized version for average link clustering [Benzécri, 1982; de Rham, 1980], which runs in $O(N^2 d)$ and needs only $O(N)$ space. They use it successfully for clustering image feature descriptors into visual vocabularies (Section 2.3).

2.2.3 Measuring Cluster Quality

When clustering data and possibly comparing several methods, the question arises, how the quality of the clustering result should be measured. There are two main options: either one uses some general statistical quality measure for the clusters (compactness *etc.*), or, if the clustering module is part of a larger processing pipeline, measure its effect on the output of the whole system.

The latter is the most pragmatic, but requires that a full system is in place and the result may be specific to that system. Furthermore, if a large range of parameters has to be varied, evaluation becomes very time consuming. On the other hand, when using a statistical measure, it can only measure certain properties, which may or may not correlate with the effect on a complete system.

As an example, we mention one statistical measure, which we also use in Chapter 3, namely the Silhouette. For a given cluster, $X_j (j = 1, \ldots, N)$, this method assigns to each sample of X_j a quality measure, $s(i) (i = 1, \ldots, m)$, known as the Silhouette width. The Silhouette width is a confidence indicator on the membership of the ith sample in cluster X_j. The Silhouette width for the ith sample in cluster X_j is defined as:

$$s(i) = \frac{a(i) - b(i)}{max(a(i), b(i))}$$

where $a(i)$ is the average distance between the ith sample and all of the samples included in X_j and $b(i)$ is the minimum average distance between the ith sample and all of the samples clustered in X_k $(k = 1, \ldots, c; k \neq j)$. From this formula it follows that $-1 \leq s(i) \geq 1$. When a $s(i)$ is close to 1, one may infer that the i-th sample has been well clustered, *i.e.* it was assigned to an appropriate cluster. When a $s(i)$ is close to zero, it suggests that the i-th sample could also be assigned to the nearest neighboring cluster. If $s(i)$ is close to 1, one may argue that such a sample has been misclassified. Thus, for a given cluster, X_j $(j = 1, \ldots, c)$, it is possible to calculate a cluster Silhouette S_j, which characterizes its heterogeneity and isolation properties:

$$S_j = \frac{1}{m} \sum_{i=1}^{m} s(i)$$

where m is number of samples in S_j.

Calculating the silhouette is computationally quite demanding for large datasets.

2.3 Image Representation with Visual Words

The bottleneck of the recognition pipeline described in Section 2.1 is often its second module, feature matching. This is due to several reasons. Firstly, if the number of images in a database is large (possibly hundred of thousands of images) determining matching features for a query feature by calculating its nearest neighbors becomes infeasible in reasonable time. Several methods for fast nearest neighbor search exist, and have also been applied to this problem (*e.g.* [Lowe, 2004] uses an optimized kd-tree). However, many of these approaches work well only in low dimensional spaces, and also suffer from another problem: finding matches for local image features is not necessarily solved best by simply finding the nearest neighbor. It is rather a bounded nearest neighbor search problem, where the bounds may depend both on the application (*e.g.* object class detection versus detection of specific objects) and even on the "meaning" of an individual feature. Some features may describe a more specific element, some of them a more general one, *i.e.* the density of a set of matching features in the vector space cannot be described with the same parameter for all features.

Thus, several works [Weber *et al.*, 2000a; Dance *et al.*, 2004; Fergus *et al.*, 2003; Leibe and Schiele, 2003] have proposed grouping features into so called *visual vocabularies*. Images are then represented as *bags of visual words*[1] or *bags of features*, *i.e.* groups or clusters of features describing the same visual primitive element. All features assigned to the same visual word are deemed matched. This representation is similar to a bag of words, used in document analysis and text retrieval.

Visual vocabularies are typically obtained by clustering the feature descriptors in high dimensional vector space. The dataset (or a sample) is clustered into k representative clusters, where each cluster stands for a visual word. The resulting clusters can be more or less compact, thus representing the variability of similarity for individual feature matches. The value of k depends on the application, ranging from a few hundred or thousand entities for object class recognition applications up to 1 million for retrieval of specific objects from large databases. This shows how the clusters are used to form a vocabulary with more or less variability in the individual visual words: in object class recognition, the individual instances of a class can have large variations, while in retrieval for specific objects very similar features have to be found.

The complete process for encoding an image with a visual vocabulary is summarized in Figure 2.2. Features are clustered into a visual vocabulary. Each feature is then

[1] a bag is the same as a multiset, *i.e.* a set where multiple occurrences of an item are considered

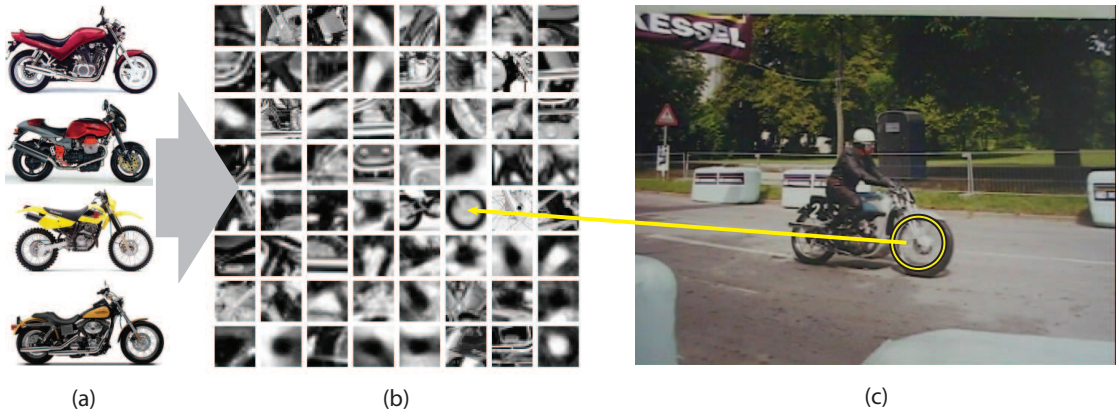


Figure 2.2: *Bag of Features approach: The features from a database of images (a) are clustered into a visual vocabulary. Each cluster is represented by an id, the visual word (b). The features of any image are then represented by the id of their closest cluster (c).*

assigned to its closest cluster. The image is now represented as a set of regions, which carry as a label the visual word id, instead of a high dimensional descriptor vector. In other words, the image can now be encoded as a histogram over the visual words appearing in the image. Matching two images now consists simply of comparing their visual word ids, or correlating their histograms. In a retrieval scenario, where a query image is compared to the images in a database, matching consists now of finding the closest visual word for each feature, instead of finding the nearest neighbor from the whole database. This is much more efficient, since typically $k << N$ for k clusters forming the visual vocabulary and N features in the database.

For image and video retrieval based on visual vocabularies often several additional methods are borrowed from text retrieval [Salton and McGill, 1986], *e.g.* the most frequent and infrequent visual words are removed from the images using a 'stop-list', or the features are ranked using a $tf * idf$ variant, weighting frequently occurring features lower.

Using visual vocabularies has been successful in video retrieval [Sivic and Zisserman, 2003] and many approaches in object class recognition [Weber *et al.*, 2000a; Agarwal and Roth, 2002; Dance *et al.*, 2004; Fergus *et al.*, 2003; Leibe and Schiele, 2003]. For clustering, most often k-Means is used, but other methods are used, too, *e.g.* [Leibe and Schiele, 2003] use a hierarchical agglomerative method, which results in better clustering results for their application.

Historically, vocabulary representations have been around for some time before they were used in combination with local features, *e.g.* for texture analysis the textons

by [Leung and Malik, 2001] or face recognition [Wiskott *et al.*, 1997]. The earliest uses for categorization or detection of "arbitrary" objects were probably the ones by [Burl *et al.*, 1998; Weber *et al.*, 2000b; Agarwal and Roth, 2002]. Many approaches have used variations of this theme since, *e.g.* [Weber *et al.*, 2000a; Fergus *et al.*, 2003; Fei-Fei *et al.*, 2003; Agarwal and Roth, 2002; Borenstein and Ullman, 2002; Feltzenswalb and Hutenlocher, 2005; Dance *et al.*, 2004; Leibe and Schiele, 2003; Sivic and Zisserman, 2003]. Various details and directions are explored in these works, e.g. various applications (retrieval or classification), types of clustering (k-means, hierarchical clustering), manual selection of primitive parts rather than clustering, determining the optimal number of clusters, investigation of the clusters in the feature spaces, *etc*.

We use variations of these visual vocabularies in many parts of this work. In Chapter 3 we mine configurations of visual words using itemset mining methods. Note, that in this context we treat images as sets of features, and not as bags of features, *i.e.* every occurrence of a visual word is only counted once. This is an approximation, which is justified by several reasons: first, multiple occurrences of a visual word often stem from non discriminative, repeated patterns. Second, we typically create a histogram of visual words from a localized neighborhood and not from the entire image. Here, considering multiple occurrences of the same feature is even less important.

In Chapter 6 we look at retrieval from large databases of local features in more detail. We use both nearest neighbor and visual vocabulary based approaches and investigate several properties of the image matching process using local features.

2.4 Frequent Itemset Mining

Frequent itemset mining is a very popular family of methods to detect the joint occurrence of certain items from a large body of data. They have their origin in market basket analysis, where large databases of customer transactions have to be analyzed to gain insights into the buying habits of shoppers. A typical desired insight could be of the form: 90% of customers who buy bread also buy milk. In a physical store this insight would allow placing certain articles next to each other to generate higher sales. In on-line stores, this enables making buying suggestions based on items already placed in the shopping basket. This particular feature is very common for book or music recommendation on platforms such as `amazon.com`.

Market basket analysis was the main application considered in the first publications on itemset mining [Agrawal *et al.*, 1993], however, the same kind of problem has been analyzed in various other contexts since. This includes web usage mining [Cooley *et al.*, 1993], robust collaborative filtering [Sandvig *et al.*, 2007], fraud detection in

on-line advertising [Metwally *et al.*, 2005], document analysis [Holt and Chun, 1999] or massive recommendation systems for related search queries [Li *et al.*, 2008a].

The remainder of this section is structured as follows: we start with a formulation of the itemset mining problem, then discuss several algorithms, which solve the problem efficiently, and finally look at some alternative quality measures for the mining results.

2.4.1 Frequent Itemset and Association Rules

Here we summarize the relevant definitions and terminology for frequent itemsets and association rules.

Let $I = \{i_1 \dots i_p\}$ be a set of p *items*. Let A be a subset of I with l items, *i.e.* $A \subseteq I, |A| = l$. Then we call A a *l-itemset*.

A transaction is an itemset $T \subseteq I$ with a transaction identifier $tid(T)$. A transaction database D is a set of transactions with unique identifiers $D = \{tid(T_1) \dots tid(T_n)\}$, $tid(T_i) \neq tid(T_j) \; \forall \; \{i, j\} \in I \mid i \neq j$.

We say that a transaction T *supports* an itemset A, if $A \subseteq T$. We can now define the support of an itemset A in the transactions-database D as follows:

Definition 2.4.1 (Support of an itemset)**.** *The support of an itemset $A \in D$ is*

$$support(A) := \frac{|\{T \in D \mid A \subseteq T\}|}{|D|} \in [0, 1]$$

Conversely, for each itemset we can also find the transactions, which support the itemset:

Definition 2.4.2 (Cover of an itemset)**.** *The cover of an itemset A in D consists of the set of transaction identifiers of transactions in D that support A:*

$$cover(A, D) := tid(T) \mid (T \in D, A \subseteq T).$$

When mining itemsets, we are interested in those sets, that occur frequently in the database:

Definition 2.4.3 (Frequent itemset)**.** *An itemset A is called frequent in D if $support(A) \geq s$ where s is a threshold for the minimal support defined by an expert.*

Two special types of frequent itemsets are also often discriminated in the literature:

Definition 2.4.4 (Closed itemset and maximal itemset)**.** *A frequent item set A is called closed if no superset has the same support. A frequent item set A is called maximal if no superset is frequent.*

After mining frequent itemsets, one is often interested in the statistical dependence between the individual items or subsets that form a set. These dependences are typically expressed in the form of association rules.

Definition 2.4.5 (Association rule)**.** *An association rule is an expression $A \rightarrow B$ where A and B are itemsets (of any length) and $A \cap B = \varnothing$.*

The quality or interestingness of a rule is typically expressed in the *support-confidence framework*, which was introduced in [Agrawal *et al.*, 1993].

Definition 2.4.6 (Support of a rule)**.** *The support of an association rule $A \rightarrow B$ is*

$$supp(A \rightarrow B) := supp(A \cup B) = \frac{|\{T \in D | (A \cup B) \subseteq T\}|}{|D|}$$

In other words, the support of a rule is the support of the joined itemsets that make up the rule. The support of a rule measures its statistical significance.

Definition 2.4.7 (Confidence of a rule)**.** *The confidence of an association rule $A \rightarrow B$ is*

$$conf(A \rightarrow B) = \frac{supp(A \cup B)}{supp(A)} = \frac{|\{T \in D | (A \cup B) \subseteq T\}|}{|\{T \in D | A \subseteq T\}|}$$

The left-hand side of a rule is called antecedent, the right-hand side is the consequent.

The confidence is a measure of the strength of the implication $A \rightarrow B$. Note that the confidence can be seen as a maximum likelihood estimate of the conditional probability that B is true given that A is true [Hand, 2001].

To get a feel for the application of these measures, let's consider a simple example:

Example 2.4.1. *The classic application for association rules is market basket data analysis. In this context, an itemset refers to a set of products. A transaction is the set of products bought by a particular customer. Consider the transactions in table 2.1. Suppose we want to find support and confidence of the famous rule $\{Diaper, Milk\} \rightarrow Beer$:*

$$support(\{Diaper, Milk\} \rightarrow Beer) = \frac{support\{Diaper, Milk, Beer\}}{|D|} = \frac{2}{5} = 0.4$$

$$confidence(\{Diaper, Milk\} \rightarrow Beer) = \frac{support\{Diaper, Milk, Beer\}}{support\{Diaper, Milk\}} = 0.66$$

TID	Items
1	Bread, Milk
2	Beer, Diaper, Bread, Eggs
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Bread, Diaper, Milk

***Table 2.1**: Example: Transactions from a store*

That means, if a customer has already diapers and milk in his shopping basket, with 66% probability he will also buy beer.

In summary, the task that itemset mining algorithms have to solve is, given a minimal support threshold s, to detect all frequent itemsets (*i.e.* $A \mid support(A) > s$) in a database D in an efficient manner. In a second step they also have to create association rules from the mined itemsets. Some algorithms which tackle this task are described in the following.

2.4.2 Frequent Itemset Mining Algorithms

The earliest algorithm to solve the itemset mining task was the APriori algorithm [Agrawal *et al.*, 1993]. Many improved algorithms have been proposed since, among them most notably FP-Growth [Han *et al.*, 2000] and ECLAT [Zaki, 2000].

The key insight to be able to perform an efficient frequent itemset search is summarized in the monotonicity property, or equivalently, the downward closure property:

Lemma 2.4.1 (Downward closure of support)**.** *Given a transaction database D, let A, B be two itemsets. Then, $A \subseteq B \rightarrow support(B) \leq support(A)$.*

Proof. This follows immediately from

$$cover(B) \subseteq cover(A)$$ □

In other words all l-subsets of frequent $(l+1)$-sets must also be frequent.

The APriori algorithm takes advantage of this property and allows us to find frequent itemsets very quickly.

APriori

The APriori algorithm is shown in Algorithm 1. The algorithm performs a breadth-first search through the search space of all itemsets by iteratively generating candidate itemsets C_{l+1} of size $l+1$. It alternates between two phases: a database

pass phase, where the support of the itemsets in C_l is calculated and checked if it surpasses the frequency threshold s, and the phase of candidate formation for $l+1$ itemsets. The main disadvantage of the APriori algorithm is that it requires multi-

Algorithm 1: APriori

```
1: l ← 1, L ← ∅
2: C_l ← {{A} | A item of size 1, A ∈ D}
3: while C_l ≠ ∅ do
4:    L_l ← ∅
5:    database pass:
6:    for A ∈ C_l do
7:       if A is frequent then
8:          L_l ← L_l ∪ A
9:       end if
10:   end for
11:   candidate formation:
12:   C_{l+1} ← sets of size l + 1 whose all subsets are frequent
13:   C_l ← C_{l+1}
14:   L ← L ∪ L_l
15: end while
16: return L
```

ple passes over the database required for the support counting procedure, thus most research towards improving performance has focused on that aspect.

The computational complexity of the algorithm can be divided by the two phases, the database pass phase and the candidate generation. As derived in [Hand *et al.*, 2001] the worst-case complexity for each iteration of the database pass phase is approximately square in L_l, where L_l is the number of frequent l-itemsets used to generate the $l+1$ stes from. In practice, however this part of the algorithm usually runs linear in L_l.

Checking a frequent set C_l for frequency (line 7 in Algorithm 1) requires testing its presence in all transactions of the database D, in each iteration the complexity is thus $O(|C_l|np)$, where C_l is the number of candidate l-itemsets, n is the number of transactions and p is the number of items.

The number of iterations of the algorithm depends on the data, with k iterations where k is the number of items in the largest frequent set.

An improved version of APriori was already proposed in [Agrawal *et al.*, 1993]. The so-called AprioriTid algorithm reduces the time needed for the support counting procedure by replacing every transaction in the database by the set of candidate itemsets that occur in that transaction. This is done repeatedly at every iteration l.

A detailed theoretical analysis of the complexity of mining frequent patterns has recently been carried out in [Yang, 2006].

FP-Growth

Unlike APriori, some algorithms such as ECLAT and FP-Growth apply a depth-first search. As an example, we describe FP-Growth (Frequent Pattern Growth) [Han *et al.*, 2000] in this section.

FP-Growth builds on two additional observations besides Lemma 2.4.1:

- Consider the $cover(A)$ of an itemset A (*i.e.* the simple fact that an itemset A is a subset of each transaction containing A). Thus, one can select transactions containing A to form a conditional database (CDB), and find patterns containing A from that conditional database $\{a, b\}, \{a, c\}, \{a\} \rightarrow \{a, b, c\}$.
- To prevent the same pattern from being found in multiple CDBs, all itemsets should be sorted by the same manner (e.g., by descending support)

The first observation allows for a divide-and-conquer approach: the conditional databases are smaller sub-problems to be solved. To that end, FP-growth uses a tree structure to store the database in a compressed form. The first step that the algorithm performs is to remove infrequent items and to sort the transactions based on the remaining items. FP-Growth then compresses these cleaned transactions into a prefix tree (the FP-tree), the root of which is the most frequent item (*i.e.* the FP tree is very similar to the prefix trees used in Huffman coding). Each path on the tree represents a set of transactions that share the same prefix; each node corresponds to one item. Each level of the tree corresponds to one item, and an item list is formed to link all transactions that possess that item. Storing transactions in the FP-tree in support descending order helps keeping the database small, since in general the more frequently occurring items are arranged closer to the root of the FP-tree and thus are more likely to be shared.

FP-Growth then starts to mine the FP-tree for each item whose support is larger than s by recursively building its conditional FP-tree. With this, the problem of finding frequent itemsets is converted to searching and constructing trees recursively.

The algorithm is outlined in Algorithm 2. It starts with initializing the data structures and then recursively the function *Growth* shown in Algorithm 3, which builds and searches the conditional trees. Further details and examples can be found in [Han *et al.*, 2000].

Algorithm 2: FP-Growth

Data: Database D,minimal support s
Result: Frequent itemsets

```
Define and clear F-List : F[ ];
foreach Transaction T_i ∈ D do
    foreach Item a_j ∈ T_i do
        F[a_j] + +;
    end
end
Sort F[ ];
Define and clear the root of FP-tree : r;
foreach Transaction T_i ∈ D do
    Make T_i ordered according to F[ ];
    ConstructTree (T_i, r);
end
foreach Item a_i ∈ I do
    Call Growth (r, a_i, s );
end
```

2.4.3 Interestingness Measures for Itemsets and Rules

Above we used the measures support and confidence to judge the quality or interestingness of frequent itemsets and association rules. However, the mining literature proposes several alternative measures, which might be more appropriate or serve simply as additional "filters", depending on the application. They include:

1. All-confidence [Omiecinski, 2003]
2. Collective strength [Aggarwal and Yu, 1998]
3. Conviction [Brin *et al.*, 1997]
4. Leverage [Piatetsky-Shapiro, 1991]
5. Lift [Brin *et al.*, 1997]
6. Normalized χ^2 [Silverstein *et al.*, 1998]
7. Difference of support/confidence quotient to 1 [Borgelt, 2003]

All these measures and many more are compared *e.g.* in [Tan *et al.*, 2002]. Another particularly interesting work is [Silverstein *et al.*, 1998] since it explores the commonalities as well as differences between correlations and associations (roughly

Procedure `Growth`
Data: r, a, s
if *r contains a single path Z* **then**
  foreach *combination γ of the nodes in Z* **do**
    Generate pattern $\beta = \gamma \cup a$ with support = minimum support of nodes in γ
    if $support(\beta) > s$ **then**
      `Output` (β)
    end
  end
end
else
  foreach *b_i in r* **do**
    Generate pattern $\beta = b_i \cup a$ with support = $support(b_i)$ **if** $support(\beta) > s$ **then**
      `Output` (β)
    end
    Construct conditional DB for β;
    Construct conditional FP-tree for β: $Tree_\beta$;
    if $Tree_\beta \neq$ **then**
      `Growth` $(Tree_\beta, \beta, s)$
    end
  end
end

spoken, associations are positive correlations). The interested reader is referred to the respective publications for further detail.

In the work at hand, besides support and confidence, we also make use of the last measure in the list above, the difference of the support quotient to 1. The reasoning for this measure is as follows: when mining itemsets, we are especially interested in sets whose items show strong dependence, or, conversely weak independence. A measure for independence can be defined as follows. Assuming perfect independence, the expected value for the support of an itemset is computed from the product of the supports of the individual items. The ratio of actual and expected support of an itemset is computed and its difference to 1 serves as an interestingness measure (*i.e.* the difference to perfect independence):

$$dep(A) = 1 - \frac{\prod_{i=1}^{l} support(A[i])}{support(A)} \tag{2.1}$$

where A is an itemset of length l and $A[i]$ is the i-th item of the itemset A. Only itemsets for which this difference is above a given threshold are then retained as interesting.

Itemset mining is one of the key techniques used in this thesis, and is applied throughout Chapter 3 and in some parts of Chapter 4.

2.5 Graph Mining

Graph mining belongs to the field of structured data mining, which, besides graphs, includes mining XML data, relational databases *etc.* Graph mining has a wide range of applications, many of them in chemical compound analysis. A large body of research has thus been published about graph mining, a good review can be found in [Washio and Motoda, 2003]. The authors categorize the approaches to graph-based data mining into five groups:

- greedy search based approaches
- inductive logic programming based approaches
- inductive database based approaches
- mathematical graph theory based approaches
- kernel function based approaches

In the work at hand we focus on mathematical graph theory based approaches, since they are conceptually close to itemset mining. For a description of the other approaches the interested reader is referred to [Washio and Motoda, 2003]. The approaches in this class have in common, that they borrow essentially the same terminology and the same search concepts from frequent itemset mining. The key measure is the (minimal) support for frequent subgraphs (where frequent subgraphs are again those, which have a support higher than a threshold s), and mining is based on candidate generation motivated by the downward closure property (Lemma 2.4.1): the subgraphs of any frequent subgraph must be frequent, too.

Algorithms from this class include *e.g.* AGM (Apriori-based Graph Mining) [Inokuchi *et al.*, 2003], FSG (Frequent SubGraph Discovery) [Kuramochi and Karypis, 2001], gSpan (graph-based Substructure pattern mining) [Yan and Han, 2002], CloseGraph [Yan and Han, 2003], and MoSS/MoFa [Borgelt and Berthold, 2002].

AGM

Just like the APriori algorithm (Section 2.4.2) for frequent itemset mining, AGM [Inokuchi *et al.*, 2003] starts from frequent one-vertex graphs and generates candidate graphs

of larger sizes by pairwise joining of frequent subgraphs that satisfy the following two conditions:

First, the frequent sub-graphs $G(\mathbf{X_k})$ and $G(\mathbf{Y_k})$ to be joined must consist of k vertices with identical elements except for those in the k-th row and k-th columns of their adjacency matrices $\mathbf{X_k}$ and $\mathbf{Y_k}$, respectively:

$$\mathbf{X_k} = \begin{pmatrix} X_{k-1} & x_1 \\ x_2^T & 0 \end{pmatrix}, \quad \mathbf{Y_k} = \begin{pmatrix} X_{k-1} & y_1 \\ y_2^T & 0 \end{pmatrix}.$$

Then the graphs are joined to form a new graph $G(\mathbf{Z_{k+1}})$ having the adjacency matrix

$$\mathbf{Z_{k+1}} = \begin{pmatrix} X_{k-1} & x_1 & y_1 \\ x_2^T & 0 & z_{k,k+1} \\ y_2^T & z_{k+1,k} & 0 \end{pmatrix},$$

where $z_{k,k+1}$ and $z_{k+1,k}$ represent an edge label between the k-th vertices of $\mathbf{X_k}$ and $\mathbf{Y_k}$.

Second, in order to avoid redundancy (the same graph can be produced by switching $\mathbf{X_k}$ and $\mathbf{Y_k}$), the two graphs may only be joined if

$$code(\text{the first matrix}) \leq code(\text{the second matrix})$$

where $code(g)$ stands for an invariant representation of a graph g – examples of invariant representations can be found in [Washio and Motoda, 2003].

Note that AGM executes a complete search and thus finds all frequent subgraphs. AGM is also capable of handling labeled vertices and edges. However, labeled edges require a conversion of the graph by inserting a special node in place of each edge label. For a dense graph, this conversion results in a graph much larger than the original one.

FSG

FSG [Kuramochi and Karypis, 2001; 2004] is similar to the AGM algorithm. However, FSG achieves higher efficiency by using graph vertex invariants (*e.g.* degree of the vertices and the labels of the vertices and edges) and keeps transaction ids (TIDs). (The latter is in fact similar to the AprioriTID algorithm for itemset mining). Using TIDs, FSG keeps for every frequent subgraph a list of the transaction identifiers that support it. This allows pruning generated candidate subgraphs before calculating costly subgraph-graph isomorphism: if the intersection of the TID lists of $G(\mathbf{X}_k)$ and $G(\mathbf{Y}_k)$ is shorter than the minimal support *minsupp* the new candidate graph created by joining $G(\mathbf{X}_k)$ and $G(\mathbf{Y}_k)$ couldn't be frequent and can

thus be pruned from the list of candidate graphs to be checked for frequency. Otherwise its frequency is computed by using a subgraph isomorphism algorithm on the limited search space determined by the intersection of the TID-lists.

Like AGM, FSG performs a complete search and thus finds all frequent subgraphs, but unlike AGM FSG is capable of handling labeled vertices and edges without a modification of the graph.

gSpan

gSpan [Yan and Han, 2002] is based on a depth-first search (DFS) and canonical labeling. The main difference to AGM and FSG is, that gSpan uses a tree representation instead of an adjacency matrix to generate an invariant code for the graphs. The frequent subgraphs are searched beginning with the frequent one-edge graphs and expanding them by one vertex at each step. gSpan relies on a special coding technique (so-called DFS codes) to encode and search the graphs. By applying this DFS coding and DFS search, gSpan can derive complete sets of frequent subgraphs over a given minimal support*s* in a very efficient manner in both computational time and memory consumption.

CloseGraph

CloseGraph [Yan and Han, 2003] is a modification of gSpan, which only finds closed frequent subgraphs. Just as in frequent itemset mining, a frequent subgraph S is called closed, if there exist no super-graphs of S with the same support. CloseGraph uses certain conditions, for a subgraph S, which — when satisfied — imply that all descendants of S are not closed and thus do not have to be considered for further expansions. As a result, the search branches of these subgraphs can be pruned completely.

Thanks to the pruning of the search branches for closed subgraphs CloseGraph outperforms gSpan in speed.

MoSS/MoFa

In contrast to the algorithms described previously, MoSS/MoFa [Borgelt and Berthold, 2002] uses a depth-first search. MoSS/MoFa starts at a graph consisting of a frequent vertex, and extends this graph iteratively by adding an edge and a vertex. The search tree is pruned using three criteria:

1. Support Based Pruning: A subtree of the search tree can be pruned, if the subgraphs belonging to this subtree have not enough support.

2. Size Based Pruning: The search tree is pruned, if a user-defined threshold for the size of the frequent subgraphs has been reached.

3. Structural Pruning: Structural pruning ensures, that every subgraph is considered only once. Structural pruning is obtained by specifying rules, on how to expand the subgraphs. These rules define an order, in which vertices and their edges can be added as well as a criterion that defines when a vertex and an edge can be added and when not.

In this work, graph mining is applied in Section 3.5, where we mine frequent subgraphs as part of an object-class recognition algorithm.

2.6 Boosting

Boosting is a technique to combine a set of weak classifiers into a strong classifier. Weak classifiers are classifiers that may be only slightly better than chance, a strong classifier should show much stronger correlation with the true classification. Thus, boosting is a meta-algorithm for supervised learning, where no specific requirements are posed on the functionality of the algorithm, except that it must behave according to the probably approximate correct learning framework (PAC) [Valiant, 1984]. There are a variety of boosting algorithms available, one of the most popular is still the early Adaboost [Freund and Schapire, 1997], which we will describe in the following.

2.6.1 Discrete Adaboost

Discrete Adaboost (or Adaboost for short – the term *discrete* is used to distinguish it from recent Adaboost variants that use real-valued classifier outputs) was introduced in [Freund and Schapire, 1997]. Boosting combines several weak classifiers $h_t(x)$ into a strong classifier $H(x)$. The weak classifiers are only required to be slightly better than chance.

Assuming the classifiers have binary $\{-1, +1\}$ output (discrete), the Adaboost strong classifier has the form

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{2.2}$$

where T is the number of weak classifiers being used (iterations), and α_t are coefficients chosen by Adaboost.

During training Adaboost assigns a weight w_i^t to each training sample i for boosting round t (with $\sum w_i^t = 1$) and calls the *weak learning algorithm* to find the best weak classifier under the given weights. Then α_t and the new weights w_i^{t+1} are calculated. This process is repeated until T boosting rounds are completed. Since none of the calculations depend on T, any other stopping criterion (like a target error rate) can be used. The learning algorithm can be summarized as follows:

- Given: labeled training samples $(x_1, y_1), \ldots, (x_n, y_n)$ with $y_i \in \{-1, +1\}$
- Initialize weights $w_i^1 = \frac{1}{N}$ for $i = 1 \ldots N$
- For $t = 1 \ldots T$
    1. Use the weak learning algorithm to find the classifier $h_t(x) \in \{-1, +1\}$ that minimizes the error $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} w_i^t$
    2. Calculate: $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
    3. Update weights: $w_i^{t+1} = w_i^t e^{(-y_i \alpha_t h_t(x_i))}$
    4. Normalize weights such that $\sum_i w_i^{t+1} = 1$
- The strong classifier is $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

In other words, AdaBoost maintains a distribution of weights for the training examples. This distribution is updated in each round - the weights of misclassified examples are increased and the weights of well-classified examples are decreased - thus, the weak learner is forced to concentrate on the difficult examples.

2.6.2 Classifier Cascades with Boosting

One of the most successful applications of boosting was for the task of real-time face detection in [Viola and Jones, 2001b]. The method is summarized in detail in Chapter 5.4. One of the main ideas of the method is to learn a cascade of classifiers which allows especially fast (real-time) recognition with the learned classifier. The idea is to reject "easy" negative matches early with little effort and then only to focus on the remaining data with more complex classifiers. For instance in [Viola and Jones, 2001b] the first classifier in the cascade - called the attentional operator - uses only two features to achieve a false negative rate of approximately 0% and a false positive rate of 40%. The effect of this single classifier is to reduce by roughly half the number of times the entire cascade is evaluated.

For learning, this means, that classifiers have to be arranged in a cascade in order of complexity, where each successive classifier is trained only on those examples which pass through the preceding classifiers. The false positives F and the detection rate

D of the entire cascade are the products of the false positives f_i and detection rate d_i of the individual stages:

$$F = \prod_{i=1}^{N} f_i \tag{2.3}$$

$$D = \prod_{i=1}^{N} d_i \tag{2.4}$$

To reach a pre-defined training goal of the whole cascade (chosen by the user), a training goal for each stage can be calculated, assuming that all stages will have about equal performance. This typically leads to a target detection rate of 0.99 and a false positive rate of 0.30 for each stage, depending on the number of stages.

Adaboost will only try to minimize the misclassification error. However by adding a constant value to the sum in equation (2.2) it is possible to increase the hit rate at the expense of the false positives. Viola and Jones have proposed the following method to train a stage:

1. Let Adaboost choose and add the next weak classifier.
2. Tune the threshold of the current strong classifier such that the desired detection rate is reached on the *validation set*.
3. If the tuned classifier does not reach the target false positive rate on the *validation set*, go back to 1.

In other words, features are added until the training goal is reached. The stage goal is the stopping criterion for Adaboost training.

2.6.3 Adaboost Variants

Since the original Adaboost publication [Freund and Schapire, 1997], many improved boosting algorithms have been proposed, for instance for real valued weak classifiers (Real Adaboost), *etc*. One extension worth mentioning is asymmetric boosting. The original Discrete Adaboost algorithm tries to minimize the number of missclassifications (this is called the *symmetric* error). However when using boosting in combination with a classifier cascade, a false positive (*e.g.* background classified as face) can still be rejected by the later stages, while a false negative (*e.g.* face classified as background) at any stage is a final decision that degrades the overall performance (equation 2.4). In other words, the cost of the classification error is not symmetric.

Motivated by the scenario of face detection, [Viola and Jones, 2001a] proposed the following solution. The authors introduced an approach to modify the weights of the training samples before each boosting round to force more attention to the positive samples. This both simplified and improved accuracy of their classifier. (Note that in that particular work Real Adaboost was used instead of Discrete Adaboost.)

We apply boosting in combination with graph mining for object class detection in Chapter 3.5 and for text detection in images of natural scenes in Chapter 5.4.

3

Frequent Itemset Mining in Visual Data

3.1 Introduction

Detection of patterns in data is probably the most important task in computer vision. It first appears at the lowest stages of a recognition pipeline (*e.g.* feature extraction), and typically reappears at every higher stage of the system. Especially at those higher stages, detection of *repeating* patterns allows us to gain valuable insights from the data. For instance, given a set of images with cars, the frequent occurrence of a certain set of local features in those images leads to the conclusion that their presence might be valuable evidence for the presence of a car in any other image.

While dealing with finding repeating patterns in data, we can discriminate two closely related tasks: data mining and learning. The discrimination of those tasks is not entirely clear in the literature, however, generally spoken in a learning environment usually a set of labeled training data to reach a certain goal is given (*e.g.* learn a model for a car), while a mining algorithm simply digs through a pile of data without a previously defined goal on what to look for (*e.g.* find the most important objects in a video sequence). Further, (machine) learning is mostly concerned with predictive models and an emphasis on performance of trained models, while data mining puts emphasis on descriptive models and patterns for existing data, and on handling large datasets.

Especially the last point, handling large dataset, was the main motivation to borrow techniques from data mining and apply them to visual data. As outlined in the introduction, the abundance of visual data available due to the rise of digital imaging devices and on-line sharing of data poses both novel opportunities and challenges for computer vision research. To handle large amounts of data, efficient algorithms are required. Specifically, we build on itemset mining algorithms as introduced

in Chapter 2.4. This choice is motivated by two factors: first, a bag of visual words (quantized local appearance features) can be described very naturally as a set. Second, itemset mining has been used successfully in a variety of domains to detect repeating combinations of items efficiently.

In this chapter, we use these algorithms first for mining tasks (find repeating patterns in existing data), but then also try to use their output for learning, i.e. as predictor for unseen data. The chapter is structured as follows: in Section 3.2 we look at a true mining task, the detection of frequently occurring objects in video data, and try to apply itemset mining to solve it. In Section 3.3 we extend the approach to tackle detection of configurations of local features as evidence for the presence of object classes. The usefulness of this evidence for object class recognition is investigated in Section 3.4 in the context of the ISM framework of [Leibe and Schiele, 2003]. Finally, Section 3.5 explores an alternative family of mining techniques namely graph mining for the same tasks.

3.2 Mining Specific Objects in Video

The goal of the method to be described in this section is to mine interesting objects and scenes from video data. In other words, to detect frequently occurring objects automatically. Mining such representative objects, actors, and scenes in video data is useful for many applications. For instance, they can serve as entry points for retrieval and browsing, or they can provide a basis for video summarization. Our approach to video data mining is based on the detection of recurring spatial arrangements of local features. The input to the mining algorithm consists of subsets of feature-codebook entries for each video frame, encoded into "transactions", as they are known in the data mining literature [Agrawal *et al.*, 1993]. We also incorporate information on spatial arrangement of features in transactions and on how to select the neighborhood defining the subset of image features included in a transaction. For scenes with significant motion, we define this neighborhood via motion segmentation. To this end, we also introduce a simple and very fast technique for motion segmentation on feature codebooks.

The remainder of this section is organized as follows. First the pre-processing steps (*i.e.* video shot detection, local feature extraction and clustering into appearance codebooks) are described. We then introduce the concepts of our mining method and show experiments on data from music video clips.

3.2.1 Shot Detection, Features and Visual Words

The main processing stages of our system rely on the prior subdivision of the video into shots. We apply the shot partitioning algorithm [Osian and Van Gool, 2004], and pick four "keyframes" per second within each shot. As in [Sivic and Zisserman, 2004], this results in a denser and more uniform sampling than when using the keyframes selected by [Osian and Van Gool, 2004]. In each keyframe we extract two types of affine covariant features (*regions*): Hessian-Affine [Mikolajczyk and Schmid, 2004b] and MSER [Matas *et al.*, 2002]. Affine covariant features are preferred over simpler scale-invariant ones, as they provide robustness against viewpoint changes. Each normalized region is described with a SIFT-descriptor [Lowe, 2004]. Next, a visual vocabulary is constructed by clustering the SIFT descriptors with an optimized hierarchical-agglomerative technique described in [Leibe and Schiele, 2003]. In a typical video, this resulted in about 8000 appearance clusters for each feature type. (Remember, that the number of clusters is determined automatically in agglomerative clustering, unlike in k-Means. The parameter that has to be set is a cut-off value for the distances used while merging clusters. Those were chosen according to the experiments shown in [Leibe and Schiele, 2003]).

We apply the 'stop-list' method known from text-retrieval and [Sivic and Zisserman, 2004] as a final polishing: very frequent and very rare visual words are removed from the codebook (the 5% most and 5% least frequent). Note that the following processing stages use only the spatial location of features and their assigned appearance-codebook id's. The appearance descriptors are no longer needed.

3.2.2 Video Mining Approach

Our goal is to find frequent spatial configurations of visual words in video scenes. For the time being, let us consider a configuration to be just an unordered set of visual words. For a codebook of size d there are 2^d possible subsets of visual words. For each of our two feature types we have a codebook with about 8000 words, which means d is typically > 10000, resulting in an immense search space. Hence we need a mining method capable of dealing with such a large dataset and to return frequently occurring word combinations.

Frequent itemset mining methods are a good choice, as they have solved analogous problems for other kinds of data, as discussed in Chapter 2.4.

Incorporating Spatial Information

In our context, the items correspond to visual words. In the simplest case, a transaction could be created for a frame, or around each feature, and would consist of an orderless bag of all other words within some neighborhood in the image. In order to include also spatial information (*i.e.* spatial locations of visual words) in the mining process, we further adapt the concept of an item to our problem. The key idea is to encode spatial information directly in the items. In each image we create transactions from the neighborhood around a limited subset of selected words $\{v_c\}$. These words must appear in at least f_{min} and at most in f_{max} frames (where f_{min} and f_{max} are parameters. This is motivated by the notion that neighborhoods containing a very infrequent word would create infrequent itemsets, neighborhoods around an extremely frequent word have a high probability of being part of clutter. Each v_c must also have a matching word in the previous frame, if both frames are from the same shot. Typically, with these restrictions, about 1/4 of the regions in a frame are selected.

For each v_c we create a transaction which contains the surrounding k nearest words together with their rough spatial arrangement. The neighborhood around v_c is divided into B sections. In all experiments we use $B = 4$ sections. Each section covers $90\,^\circ$ plus an overlap $o = 5\,^\circ$ with its neighboring sections, to be robust against small rotations. We label the sections $\{tl, tr, bl, br\}$ (for "top-left", "top-right", etc.), and

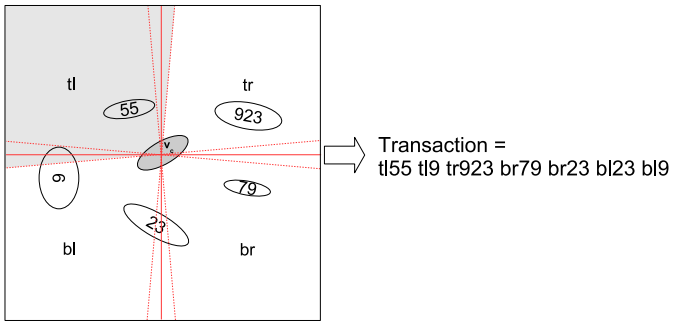


***Figure 3.1**: Creating transaction from a neighborhood. The area around a central visual word v_c is divided into sections. Each section is labeled (tl, tr, bl, br) and the label is appended to the visual word ids.*

append to each visual word the label of the section it lies in. In the example in Figure 3.1, the transaction created for v_c is $T = \{tl55, tl9, tr923, br79, br23, bl23, bl9\}$. In the following, we refer to the selected words $\{v_c\}$ as *central* words. Although the approach only accommodates for small rotations, in most videos objects rarely appear in substantially different orientations. Rotations of the neighborhood stemming from perspective transformations are safely accommodated by the overlap o. Although augmenting the items in this fashion increases their total number by a factor B, no changes to the frequent itemset mining algorithm itself are necessary. Besides, thanks to the careful selection of the central visual words v_c, we reduce the number of transactions and thus the runtime of the algorithm.

Exploiting Motion

Shots containing significant motion[1] allow us to further increase the degree of specificity of transactions: if we had a rough segmentation of the scene into object candidates, we could restrict the neighborhood for a transaction to the segmented area for each candidate, hence dramatically simplifying the task of the mining algorithm. In this case, as the central visual words v_c we pick the two closest regions to the center of the segmented image area. All other visual words inside the segmented area are included in the transaction (Figure 3.3).

We propose a simple and very fast motion segmentation algorithm to find such object candidates. The assumption is that interesting objects move independently from each other within a shot. More precisely, we can identify groups of visual words which translate consistently from frame to frame. The grouping method consists of two steps:

[1]Since shot partitioning [Osian and Van Gool, 2004] returns a single keyframe for static shots and several keyframes for moving shots, we can easily detect shots with significant motion.

Step 1. Matching words. A pair of words from two frames $f(t), f(t+n)$ at times t and $t+n$ is deemed matched if they have the same visual word ids (*i.e.* they are in the same appearance cluster), and if the translation is below a maximum translation threshold t_{max}. This matching step is extremely fast, since we rely only on cluster id correspondences. In our experiments we typically use $t_{max} = 40$ pixels and $n = 6$ since operating at four keyframes per second.

Step 2. Translation clustering. At each timestep t, the pairs of regions matched between frames $f(t)$ and $f(t+n)$ are grouped according to their translation using k-means clustering. In order to determine the initial number of motion groups k, k-means is initialized with a leader initialization [Webb, 2002], on the translation between the first two frames. For each remaining timestep, we run k-means three times with different values for k, specifically

$$k(t) \in \{k(t-1)-1, k(t-1), k(t-1)+1\} \tag{3.1}$$

where $k(t-1)$ is the number of motion groups in the previous timestep. This prevents the number of motion groups from changing abruptly from frame to frame. Furthermore, $k(t)$ is constrained to be in [2...6]. To further improve stability, we run the algorithm twice for each k with different random initializations. From the resulting different clusterings, we keep the one with the best mean silhouette value [Kaufman and Rousseeuw, 1990]. We improve the quality of the motion groups with the following filter. For each motion group, we estimate a series of bounding-boxes, containing from 80% progressively up to all regions closest to the spatial median of the group. We retain as bounding-box for the group the one with the maximal density $\frac{number\ of\ regions}{bounding\ box\ area}$. This procedure removes from the motion groups regions located far from most of the others. These are most often mismatches which accidentally translate similar to the group.

The closest two visual words to the bounding box center are now selected as the central visual word v_c for the motion group. Figure 3.2 shows detected motion groups for a scene of a music videoclip.

3.2.3 Mining an Entire Video

We quickly summarize the processing stages from the previous sections. A video is first partitioned into shots. For rather static shots we create transactions from a fixed neighborhood around each central word. For shots with considerable motion, we use as central words the two words closest to the spatial center of the motion group, and create two transactions covering only visual words within it. For frequent itemset mining itself we use an implementation of APriori from [Borgelt, 2003]. We mine Maximal Frequent Itemsets and only sets with four or more items are kept.

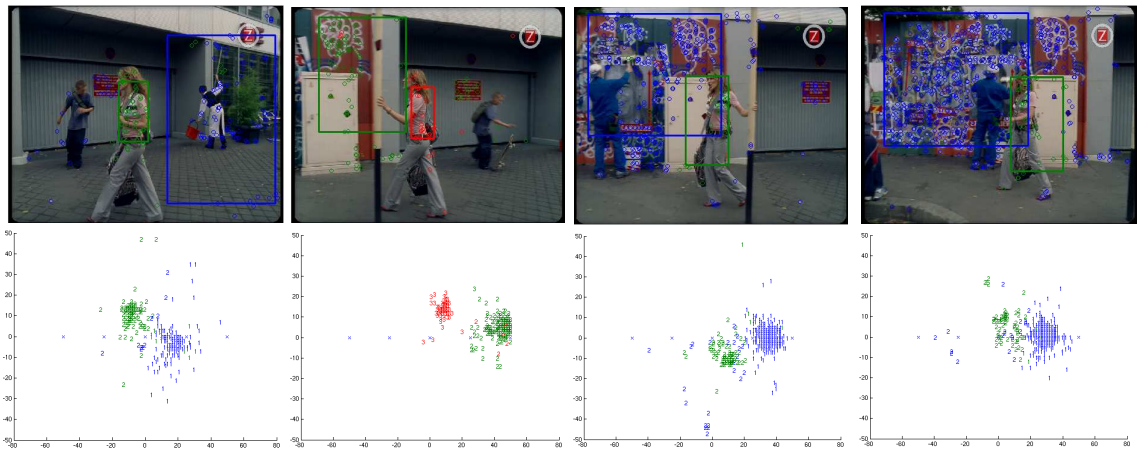


***Figure 3.2**: First row: motion groups (only region centers shown) with bounding boxes. Second row: motion groups in translation space. Note: colors do not necessarily correspond along a row, since groups are not tracked along time.*

Note how frequent itemset mining returns sparse but discriminative descriptions of neighborhoods. As opposed to the dot-product of binary indicator vectors used in [Sivic and Zisserman, 2004], the frequent itemsets show *which* visual words cooccur in the mined transactions. Such a sparse description might also be helpful for efficiently indexing mined objects.

Choosing a Support Threshold

The choice of a good minimal support threshold s in frequent itemset mining is not easy, especially in our untraditional setting where items and itemsets are constructed without supervision. If the threshold is too high, no frequent itemsets are mined. If it is too low, too many (possibly millions) are mined. Thus, rather than defining a fixed threshold, we run the algorithm with several thresholds, until the number of frequent itemsets falls within a reasonable range (usually set to more than 100 and less than 100′000 sets). We achieve this with a binary split search strategy. Two extremal support thresholds are defined, s_{low} and s_{high}. The number of itemsets is desired to be between n_{min} and n_{max}. Let n be the number of itemsets mined in the current step of the search, and s be the corresponding support threshold. If the number of itemsets is not in the desired range, we update s by the following rule and rerun the miner:

$$s^{(t+1)} = \begin{cases} s^{(t)} + \frac{(s_{high} - s^{(t)})}{2}, \; s_{low} = s^{(t)} & \text{if } n > n_{max} \\ s^{(t)} - \frac{(s^{(t)} - s_{low})}{2}, \; s_{high} = s^{(t)} & \text{if } n < n_{min} \end{cases}$$

Since the mining algorithm is very fast, we can afford to run it several times (runtimes reported in the result section).

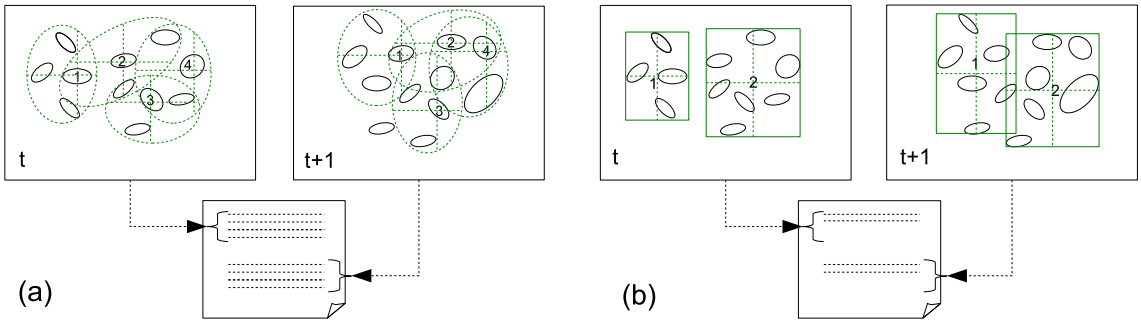


***Figure 3.3**: Creating transactions: (a) static shots: transactions are formed around each v_c from the k-neighborhood. (b) shots with considerable motion: a motion group is the basis for a transaction, thus the number of items in a transaction is not fixed but given by the size of the motion group. With (b) in general fewer transactions are generated.*

Finding Interesting Itemsets

The output of the APriori algorithm is usually a rather large set of frequent itemsets, depending on the minimal support threshold. Finding *interesting* item sets (and association rules) is a much discussed topic in the data mining literature, as outlined in Section 2.4, where we discussed several approaches, which define interestingness with purely statistical measures. For instance, itemsets whose items statistically dependent are interesting. We thus applied the measure defined from equation 2.1, which had in general a positive effect on the quality of our mining results.

Another strategy is to rely on domain-specific knowledge. In our domain, itemsets which describe a spatial configuration stretching across multiple sections tl, tr, bl, br are interesting. These itemsets are less likely to appear by coincidence and also make the most of our spatial encoding scheme, in that these configurations respect stronger spatial constraints. The number of sections that an itemset has to cover in order to be selected depends on a threshold $n_{sec} \in \{1 \dots 4\}$. Selecting interesting itemsets with this criteria is easily implemented and reduces the number of itemsets drastically (a typical value is $n_{sec} = 2$, we observed reduction of sets by a factor of about 10 to 100).

Itemset Clustering

Since the frequent itemset mining typically returns spatially and temporally overlapping itemsets, we merge them with a final clustering stage. Pairs of itemsets which jointly appear in more than F frames and share more than R regions are merged. Merging starts from the pair with the highest sum $R+F$. If any of the two itemsets

in a pair is already part of a cluster, the other itemset is also added to that cluster. Otherwise, a new cluster is created.

3.2.4 Experiments and Results

We present results on two music videos from Kylie Minogue [Minogue and Gondry, 2002; Minogue and Shadforth, 2001]. In particular the clip "Come into my world" [Minogue and Gondry, 2002] makes an interesting test case for mining, because the singer passes by the same locations four times, and she even appears replicated several times in later parts of the clip. (Figure 3.4, bottom row). Hence, we can test whether the miner picks up the reappearing objects. Furthermore, the scene gets more and more crowded with time, hence allowing to test the system's robustness to clutter.

A few of the objects mined from the 1500 keyframes long clip "Come into my world" [Minogue and Gondry, 2002] are shown in Figures 3.4 through 3.6. The full miner was used, including motion grouping and itemset filtering with $n_{sec} = 2$. The Figures show the most dominant objects that were mined. In Figure 3.4 one of the main locations of the clip is identified as important. The four rows show keyframes from each of the singer's walks through the location. Note how the "multiplication" of the main character leads to strong occlusion effects, especially in the fourth walkthrough shown on the last line. Also note the viewpoint changes. All instances of the location are mined in spite of these challenges.

Figure 3.5 shows the main character of the clip mined due to the pattern on her clothes. The character is mined in all locations (including the one from Figure 3.4) and in varying poses. However, some of the replicated instances of the singer are missed by our algorithm (3rd and 4th rows).

Figure 3.6 shows a third mined object, this time again representing one location of the video. The four rows of the Figure show again frames from each pass through the location. However, here only two frames from the last walk through the scene could be mined (the missing frames are represented by the placeholders in the fourth row). This is probably due to the increasing background occlusion throughout the video, which can be observed in the second column of the figure. Such missing frames could be recovered by adding an object-level tracking, connecting gaps between keyframes.
Figure 3.8 shows typical results for mining with a fixed 40-neighborhood, *i.e.* without motion segmentation, akin to what has been proposed by [Sivic and Zisserman, 2004]. As can be seen in subfigures 3.8a and 3.8b, only smaller parts of the large objects from Figures 3.4- 3.6 are mined. More examples of objects mined at the 40-neighborhood scale are shown in the other subfigures. Comparing these results to those in Figure 3.4 highlights the benefits of defining the neighborhood for mining based on motion segmentation. Thanks to it, objects can be mined at their actual

***Figure 3.4**: Results for clip "Come into my World" using motion segmentation. First mined cluster, a walk through the scene is shown on each line by representative keyframes.*

size (number of regions), which can vary widely from object to object, instead than being confined to a fixed, predefined size. Additionally, the singer was not mined when motion segmentation was turned off.

Figure 3.7 shows example objects mined from the clip [Minogue and Shadforth, 2001] with a 40-neighborhood. The results are less impressive than ones obtained on the clip [Minogue and Gondry, 2002]. One reason is the very dynamic nature of that particular clip, with many short shots and little translational motion, which does not result in benefits when applying our simple motion segmentation stage. Furthermore, our algorithm is naturally challenged by sparsely textured, non-rigid objects. As an example one could mention the legs of the main character. There are few features to begin with and the walking motion strongly changes the configuration of those, thus not the whole body is detected as object.

In Table 3.1 we compare quantitatively mining with motion segmentation, and with a fixed 40-neighborhood for the clip "Come into my world" [Minogue and Gondry, 2002]. Note that there are only 8056 transactions when using motion segmentation,

***Figure 3.5**: Results for clip "Come into my World". The second mined cluster representing the main character of the clip. It is mined throughout the clip in varying locations and poses*

compared to more than half a million when using a fixed 40-neighborhood. While the runtime is very short for both cases, the method is faster for the 40-neighborhood case, because transactions are shorter and only shorter itemsets were frequent. Additionally, in the 40-NN case, the support threshold to mine even a small set of only 285 frequent itemsets has to be set more than a factor 10 lower. The mean time for performing motion segmentation matching + k-Means clustering) was typically about $0.4s$ per frame, but obviously depends on the number of features detected per frame. In conclusion, we showed that our mining approach based on frequent itemsets is a suitable and efficient tool for video mining. Restricting the neighborhood by motion grouping has proven to be useful for detecting objects of different sizes at the same time.

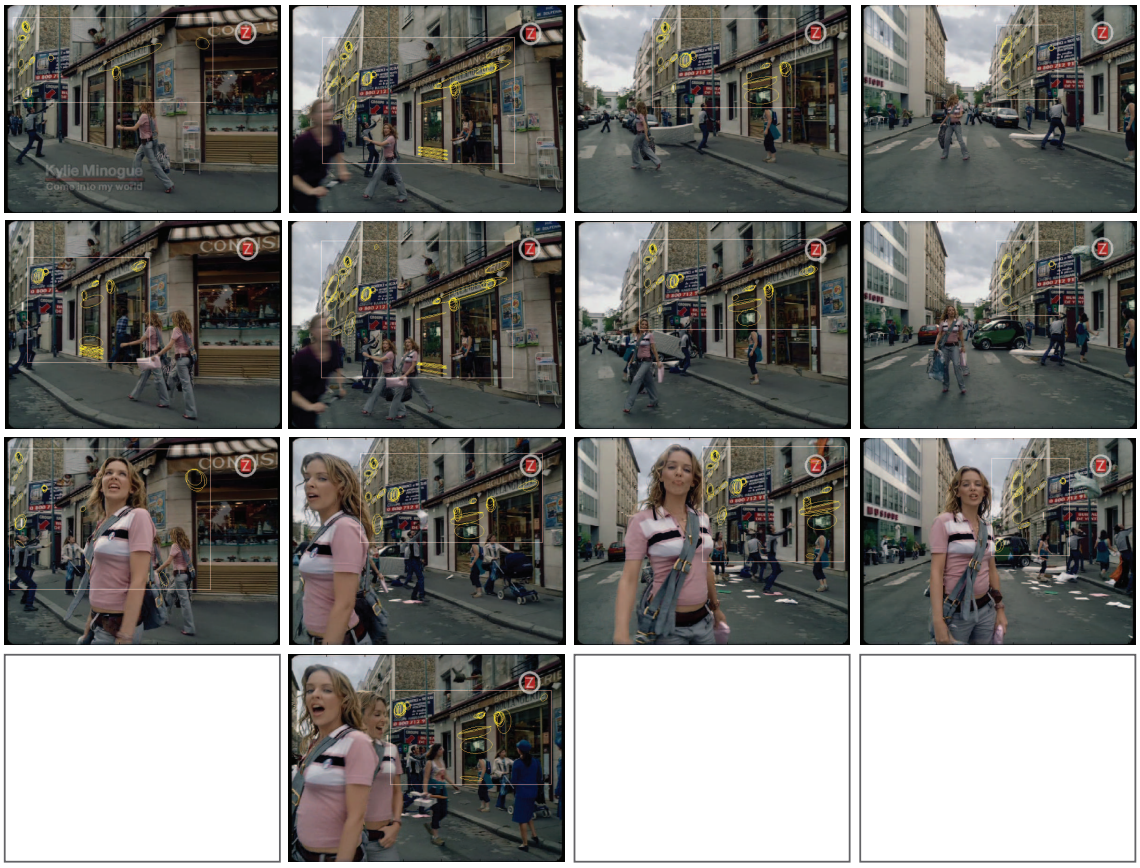


Figure 3.6: *Results for clip "Come into my World". Third mined cluster, a walk through the scene on each line is shown. Only few frames of the last pass through the location could be mined in this case (last line).*

Figure 3.7: *Results for clip "Can't get you out of my head".*

Figure 3.8*: Examples for the clip Come into my World mined at a fixed 40 neighborhood.*

Method	Regions	$\#T$	t FIMI	s	# FI	# FIF (n_s)	Cl (F,R)
M.-Seg.	$2.87 * 10^6$	8056	56.12s	0.015	27654	308 (2)	11 (2,2)
40-NN	$2.87 * 10^6$	511626	18.79s	0.0001	285	285 (0)	55 (2,2)

Table 3.1*: Motion Segmentation and 40-NN mining methods compared.* Regions*: number of regions in entire video.* #T*: number of transactions.* t FIMI*: runtime of frequent itemset mining.* s*: support threshold.* #FI*: number of frequent itemsets.* FIF*: number of FI after filtering step with n_s sections.* Cl*: number of clusters for itemset clustering with parameters F,R.*

3.3 Mining Frequent Feature Configurations

In the preceding section of this chapter we described an approach to mine frequently occurring objects from video data using itemset mining on quantized local features. The objects we were dealing with were specific objects, that is a specific person or scene were the output of the mining algorithm. Local features are also at the heart of the most successful approaches to object *class* detection and image classification [Agarwal *et al.*, 2004; Dalal and Triggs, 2005; Dance *et al.*, 2004; Feltzenswalb and Hutenlocher, 2005; Fergus *et al.*, 2005; Leibe *et al.*, 2005; Sivic and Zisserman, 2004; Opelt *et al.*, 2006]. After learning a class model from training images, these methods are capable of detecting whether a novel object instance is present in a previously unseen test image. Several recent methods go even a step further by *localizing* novel objects up to a bounding-box [Agarwal *et al.*, 2004; Dalal and Triggs, 2005] or their segmentation and outlines [Shotton *et al.*, 2006; Leibe *et al.*, 2005]. These methods are robust to clutter, scale changes, and missing object parts - properties which stem from the advantageous characteristics of local features. However, these advantages come at a price. The local feature extractor is run beforehand and without prior knowledge of the object class. As a result, on a typical image it returns a large number of features, of which only some fraction lie on the object of interest. Especially when the object appears small in the image, the total set of features has a low signal-to-noise ratio. This imposes a great burden on object detectors and other higher-level processes, as they have to find their way to the object through a sea of background features.

In this chapter we propose a mining-based method to filter this large mass of features. It selects features which have high probability of lying on instances of the object class of interest. Our technique is intended as an intermediate layer between feature extraction and object class detection. The filtered set of features our method delivers can then be fed into a higher-level object detector. Thanks to this, it starts from a much higher signal-to-noise ratio, and its performance is likely to improve. We expect our method to lead to lower false-positive rates, and possibly also higher detection rates. Besides, starting from a cleaner set of features is likely to ease other tasks as well, such as segmenting objects from the background, or determining their pose. The method's input is a set of positive training images, containing different instances of the object class, and a set of negative background images. We organize local features in semi-local neighborhoods and express these in a way suitable for data mining. We adopt again Frequent Itemset Mining, which *efficiently* analyzes the large set of all neighborhoods and returns spatial configurations of local features frequently re-occurring over the training images. From these frequent spatial configurations we now also collect discriminative Association Rules. These rules infer the presence of the object in positive images with high confidence and fire only rarely on background images. Figure 3.9 shows two typical feature configurations and the

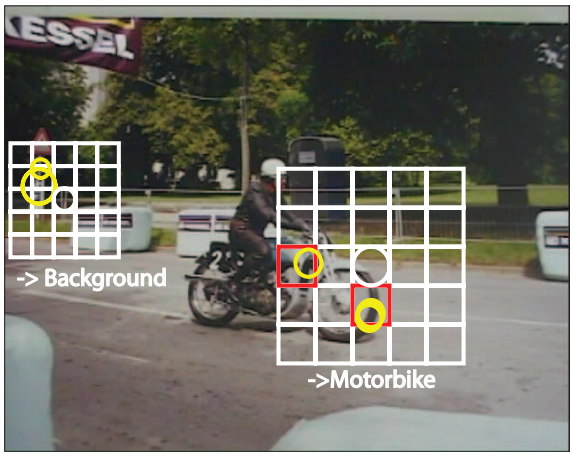


***Figure 3.9**: Example of mined rules: on the left a frequent configuration which infers background, on the right a configuration which infers the object motorbike.*

corresponding rules produced by our miner. One rule infers the presence of the motorbike, while the other corresponds to a feature configuration mined from the background. When given a novel image, we first match the mined configurations to it, and then we associate a confidence value to each feature expressing how likely it is to lie on an instance of the object class. This is obtained by accumulating the activation scores of all matched configurations involving the feature.

This approach has several advantages. First of all, the mining algorithm is designed for scalability and allows to process large training sets rapidly. Moreover, the set of rules collected from the data in this fashion are discriminative and easy to interpret. Indeed, by considering spatial configurations of neighboring features we gain higher discriminative power compared to individual features. A single local feature, even from an informative configuration, might not be distinctive enough and could occur frequently also on the background. In addition, the rules often capture configurations of local features corresponding to semantic object parts, such as motorbike wheels (Figure 3.11). The per-feature confidence values produced by our approach effectively prune away the majority of background features, and therefore act as a valuable focus-of-attention mechanism for the benefit of subsequent object detectors, e.g. [Agarwal *et al.*, 2004; Fergus *et al.*, 2005; Leibe *et al.*, 2005].

Also note, that unlike in the video mining work presented in the preceding section, we have no motion cues, and thus can't rely on a motion segmentation to identify neighborhoods to create transactions from. Hence, we present an extended and refined method for including spatial arrangement of features in the itemset mining process, which also works for the kind of data we are confronted with now: unsorted images containing instances of an object class, instead of an ordered sequence of images showing a specific object.

The remainder of this section is organized as follows. First we describe our approach to mining frequent spatial configurations of local features from training images. In subsection 3.3.2 we determine the confidence that features appearing in new images cover an instance of the object class. Finally, an extensive experimental evaluation is carried out, demonstrating our approach primes features lying on class instances and discards background ones.

3.3.1 Frequent Feature Configurations

Our technique for mining frequent feature configurations can be summarized as follows. The training set is composed of positive images, containing object instances annotated by a bounding-box, and of negative images, which do not contain any instance of the class of interest. First, a large number of spatial configurations of local image features are collected from all training images. An efficient mining algorithm is then used to select frequently occurring configurations from this large set. The next step transforms these frequent spatial configurations into association rules. These rules are built by selecting frequent spatial configurations which imply the presence of the object class with high confidence, while at the same time are discriminative against clutter (*i.e.* they occur rarely on the negative images or on non-object areas of the positive images).

These *discriminative rules* are the building blocks for a generating class-specific confidence values for features of novel images. These convey the probability that each feature belongs to an instance of the object class (Section 3.3.2).

The itemset mining algorithm is the same as in the previous section about video mining. However, now we also form association rules from the mined itemsets. Association rules have several desirable properties. Thanks to the efficient frequent itemset mining method they can be extracted even from very large bodies of data. The rule notation is easily interpretable and can be used to gain global insights into large datasets or can be analyzed by experts. These properties have led to their application in several fields such as web usage mining [Cooley *et al.*, 1993] or document analysis [Holt and Chun, 1999].

The lowest layer of our system is again built on a set of local features extracted in each image. We use a Difference of Gaussian (DoG) detector to extract regions and the SIFT descriptor [Lowe, 2004] to describe their appearance. The SIFT feature vectors are clustered into a visual vocabulary with hierarchical agglomerative clustering, just like in the preceding section.

In order to cope with the inherent uncertainty of the unsupervised clustering process, we *soft-match* each feature by assigning it to all codebook clusters whose center c is

closer than a distance threshold d_{min}. This yields a description of each region R_i by a set of codebook labels

$$\zeta_i = \{c_j \mid d(R_i, c_j) < d_{min} \ , \ j \in 1 \dots N\} \tag{3.2}$$

where N is the total number of appearance clusters.

Neighborhood-based Image Description

The second layer of our system builds an image representation from the codebook labels. The simplest representation would be a global histogram, *i.e.* a *bag of features* as discussed in Section 2.3. However, we aim at unsupervised mining and at learning useful representations for object classes. In this setting, a more informative description is necessary. Encoding not only the presence of visual words, but also their spatial arrangement yields a much stronger descriptor. Thus, we describe each image as a set of semi-local neighborhoods.

Several methods have been proposed to sample spatial neighborhoods from an image. In [Dalal and Triggs, 2005] a sliding-window mechanism samples windows at fixed location and scale steps, followed by a spatial tiling of the windows. The very different approach [Sivic and Zisserman, 2004] defines a neighborhood around each region R_c. This is represented as the unordered set of the k nearest regions, without storing any spatial information (*k-neighborhoods*).

Our approach tries to combine the best of both. We rely on the sampling of the feature extractor to define the locations R_c of the neighborhood centers. However, instead of using a k-neighborhood we use the scale of the central region R_c to define the size of the neighborhood. More precisely, all regions falling within a square of side proportional to the scale of R_c are inside the neighborhood. Subsequently, each neighborhood is split into Q tiles as shown in Figure 3.10a. For each tile we create an activation vector indicating which visual words it contains[2]. The resulting Q activation vectors are concatenated to form the neighborhood descriptor: a $(N * Q)$-dimensional sparse binary vector. Figure 3.10b shows a neighborhood descriptor for $N = 10$ and $Q = 9$. Note how in this example the top-left region is soft-matched to appearance clusters 2 and 5. The activation vector can equivalently be written as a list of non-zero indices – or, in itemset mining terminology, as a *transaction* (figure 3.10c). Note how neighborhoods can be made rotation invariant by aligning the tile grid with the dominant orientation of R_c. In otherwords, the neighborhood description is a generalized version of the neighborhood with only 4 tiles used in the previous section. Since we form a neighborhood for every region in every training image, this results in a very large number of neighborhoods (or transactions). The

[2]We do not count multiple occurrences of the same visual word in a particular tile, *i.e.* we work with sets instead of bags.

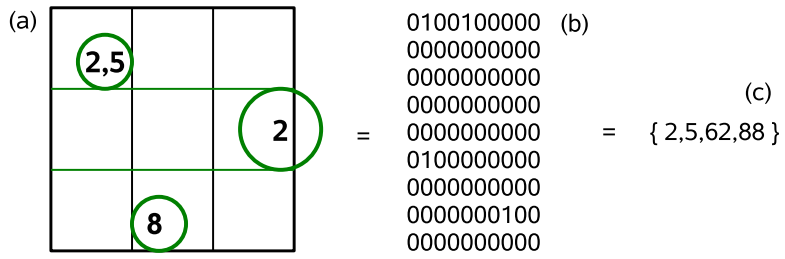


Figure 3.10: *(a) An example neighborhood with 9 tiles and 10 appearance clusters. Circles represent local features, and numbers indicate the appearance cluster(s) they are assigned to. (b) Activation vector. (c) Transaction.*

training sets in section 4.6 have between 26′000 and 74′000 transactions. Note that itemset mining can handle these amounts of data with ease – in a recent parallel implementation of FP-Growth [Li *et al.*, 2008a] datasets with 15′000′000 transactions and 85′000′000 items were mined successfully.

Mining Frequent and Distinct Configurations

Equipped with the tools introduced in the previous sections, we can now find frequent configurations of visual words efficiently. We are especially interested in mining *distinctive* configurations, which appear frequently on the object and rarely on the background.

As discussed above, each neighborhood is described by a list of non-zero indices, and generates a transaction. The input to the mining algorithm is the database containing all transactions. In order to discriminate against background data, we add transactions from the negative training set to the database. All transactions originating from instances of the object class are assigned the label "object" as an additional item, while we append the item "background" to background transactions. For example, the complete transaction for the neighborhood in figure 3.10 is $\{2, 5, 62, 88, object\}$ (assuming it lies on an object).

We run the APriori [Agrawal *et al.*, 1993] algorithm on the transaction database in order to mine frequent itemsets and association rules. We filter the resulting rules to keep only those which infer the object label with high confidence, *i.e.*

$$conf(\mathcal{C} \rightarrow object) > conf_{min} \tag{3.3}$$

where the antecedent $\mathcal{C}$ is a frequent configuration and $conf_{min}$ is a confidence threshold. Notice how a rule does *not* have a high confidence if it appears frequently on both objects and background. This can be understood by inspecting Definition (2.4.7), where confidence expresses the strength of the implication $\mathcal{C} \rightarrow$ object (see

section 2.4). Hence, our approach finds frequent *and* distinctive feature configurations. Moreover, frequent itemset mining finds these prototypical configurations very efficiently from the immense search space of all 2^{N*Q} possible configurations (typically $N \simeq 3000$ and $Q \simeq 16$).

As additional advantage, many of the mined rules have semantic qualities, as shown in Figure 3.11. The top left image shows activations of one particular rule on the Caltech-4 set [Fergus *et al.*, 2003] used to mine rules for motorbikes. Activations on two novel test images are shown in the second and third row (see next section for how to match the mined configurations to new images). The regions matching the antecedent $\mathcal{C}$ of the rule are marked in yellow. The central region R_c defining the neighborhood $\mathcal{P}$ is shown in white[3]. Notice the variability in the shape and appearance of the motorbikes, and the different scales of the neighborhoods (automatically adapting to the image data). The rule in the figure is $\{32909, 34622, 46292\} \rightarrow motorbike$ with $s = 3\%$ support and $c = 100\%$ confidence. This rule is one of the most discriminant found for *motorbike*. This makes sense, as wheels are its most characteristic parts. Similar observations can be made for the giraffes in the right column.

3.3.2 Class-specific Feature Confidence

The frequent feature configurations $\mathcal{C}$ mined from the neighborhoods in the training images represent frequent and discriminant fragments of an object class. They describe neighborhoods characteristic for the object class.

Given a new test image, we can now match the mined configurations to it, and hence discover features lying on instances of the object class. To achieve this, we start by generating all neighborhoods $\mathcal{P}$ of the new image (one for each region, as described in section 3.3.1). Every mined configuration $\mathcal{C}$ is now matched to each image neighborhood $\mathcal{P}$ as follows. A configuration can be written as a sparse activation vector. Hence, the test image neighborhoods can be matched efficiently by a sparse dot-product:

$$m(\mathcal{C}, \mathcal{P}) = \begin{cases} 1 & \text{if } \mathcal{C} * \mathcal{P} = |\mathcal{C}| \\ 0 & \text{if } \mathcal{C} * \mathcal{P} \neq |\mathcal{C}| \end{cases} \tag{3.4}$$

where $|\mathcal{C}|$ is the number of features in $\mathcal{C}$, and $m(\mathcal{C}, \mathcal{P}) = 1$ indicates a match. In other words, a frequent configuration $\mathcal{C}$ matches a candidate neighborhood $\mathcal{P}$ if their dot product equals the number of visual words in $\mathcal{C}$.

From matched neighborhoods of the test image we can derive a measure of the probability for a feature to lie on an instance of the object class. This measure effectively enables to pre-select features lying on the object, and hence it can substantially ease the life of a subsequent object detector. Thanks to this, the latter

[3] R_c is not part of the rule. In this example the rule consists of the yellow regions only.

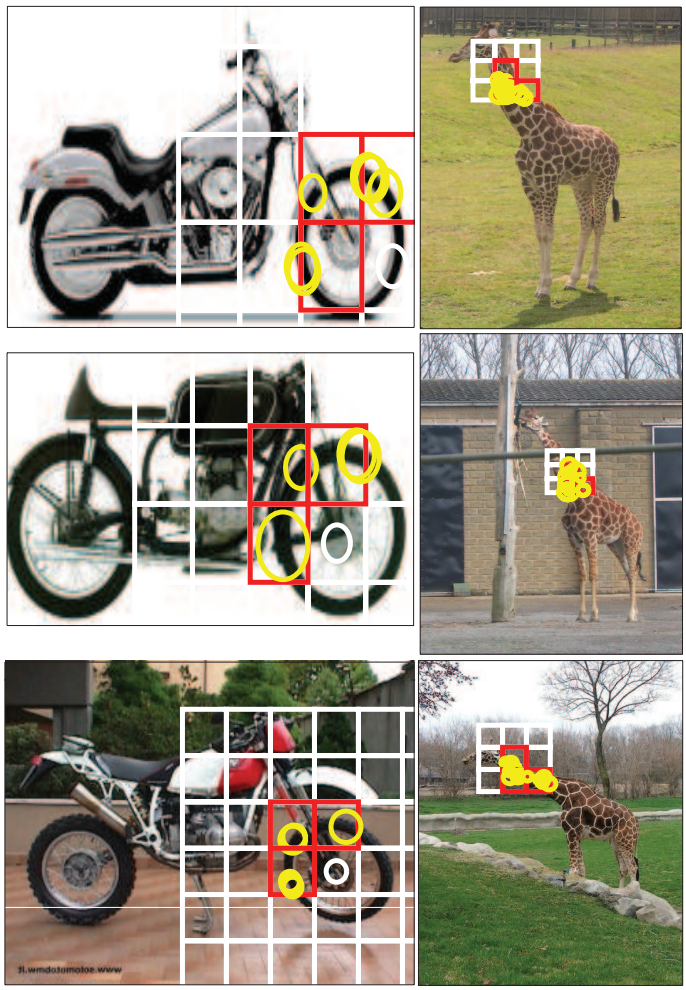


Figure 3.11: *Discriminant Frequent Spatial Configurations. First row: examples of activations on the training set. Second/third row: examples of activations on the test-set. Note: R_c (white) is not part of the mined rule in this example.*

can focus on higher level tasks, such as localizing the object up to a bounding-box, determining its precise extent (outlines), its pose, a part decomposition, and so on. We compute this class-specific feature confidence measure as follows. For each feature in the image, we count how often it is part of a matched neighborhood. The more matched configurations a features participates into, the more it is likely to cover part of an object instance. More precisely, the confidence measure for each feature R_i is defined as:

$$conf(R_i) = \frac{1}{M * W} \sum_{\mathcal{C}} \sum_{\{\mathcal{P}|R_i \in \mathcal{P}\}} \frac{1}{k} * m(\mathcal{C}, \mathcal{P}) \tag{3.5}$$

where M is the number of configurations mined on the training data, W is the number of neighborhoods in the test image, k is the number of appearance clusters to which R_i was soft-assigned (equation (3.2)).

3.3.3 Experiments and Results

We present results on four diverse object classes. After discussing the quality of the results via some visual examples, we perform a quantitative performance evaluation. The experiments are conducted on the following datasets. The objects in the positive training images were annotated by a bounding-box, except for the *TUD Motorbikes* where full images without bounding box were used for training.

ETHZ Giraffes. Training was conducted on 93 images of giraffes we downloaded from Google Images. No background training data was used in this case. The positive test images are the 87 Giraffes from the ETHZ Shape Classes dataset [Ferrari *et al.*, 2006]. All 168 images of the other classes from [Ferrari *et al.*, 2006] are used as negative test set (as done for object detection from hand-drawings by [Ferrari *et al.*, 2006]).

GRAZ Bikes. All training data and the positive test set are as defined in the paper which originally proposed this dataset [Opelt *et al.*, 2003]. As negative test set we took the first 200 images from the CALTECH-101 background [Fei-Fei *et al.*, 2004] class. This negative test set is used as well with all following datasets.

TUD Motorbikes. The TUD Motorbikes dataset [Various, 2005] consists of 115 images containing 125 motorbikes, which we used as positive test set. The positive training images are the Caltech-4 motorbikes [Fergus *et al.*, 2003] (no bounding-boxes given). As background training set we randomly picked 200 images from the CALTECH-256 [Griffin *et al.*, 2007] background class.

CALTECH Cars Rear. This dataset features 126 rear-views of cars and 1155 street scenes without cars, used as training set. Moreover, the dataset also provides a test set of 526 images containing cars, as described in [Fergus *et al.*, 2003].

The first three datasets are particularly challenging, as objects appear in severely cluttered images, and present scale and intra-class variations. Moreover, the GRAZ Bikes and TUD Motorbikes are partially occluded in several images. The CALTECH Cars are somewhat easier, in that they appear rather centered in the images and vary only moderately in scale.

Visual Examples

We present here visual examples to demonstrate the quality of the mined feature configurations, and of features selected based on the confidence values our approach

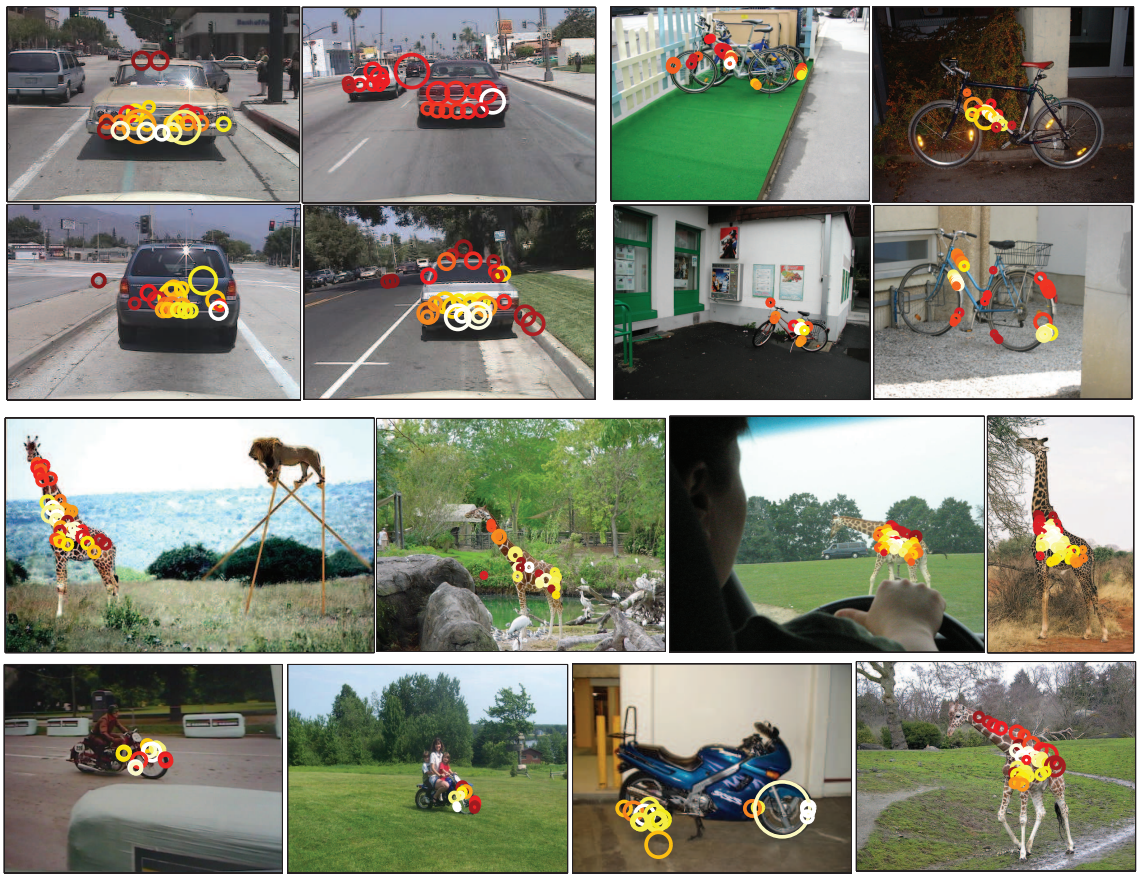


***Figure 3.12**: Results: Visual Examples. (See text for discussion.)*

delivers. Figure 3.12 shows several test images, with all overlaid features having a confidence (equation 3.5) above 20% of the maximum possible value. These features belong to configurations deemed frequent and discriminative by our method. The brighter the color of a feature, the higher its confidence.

The large majority of features are systematically selected on the object, in spite of scale changes, clutter, and intra-class variations. It is particularly interesting to notice how the selected features adapt to the class so as to cover its most discriminative parts. For bikes, the rather structural configurations of frame parts and wheel fragments dominate, whereas for giraffes the pattern of the fur is selected (*i.e.* the miner adapts to behave like a texture detector). Besides, notice how our measure effectively selects object features, and discards background ones. These results confirm that our approach effectively primes object features while pruning away the majority of background ones. Hence, it is a valuable intermediate step before applying higher-level processing such as object localization algorithms.

Quantitative Evaluation of Feature Selection

We quantify the performance of our method for assigning class-specific confidences to features, based on two experiments. In the first experiment we measure *bounding box hit rate* (BBHR) over the positive test sets. A bounding-box hit is counted if more than k features selected by our method lie on the object (inside the bounding box). Hence, BBHR is the number of BBH divided by the total number of object instances in the positive test set. To perform this evaluation we use ground-truth bounding-box annotations available for the test images (these were not used to produce the results). The rationale behind the BBHR measure is that the later processes our method is intended to aid, need at least a certain number of features to operate reliably (e.g recognition - deciding whether the object is actually present in the image, or localization - determining a bounding-box framing the object). We set BBHR in relation with the false positive rate (FPR). This is the number of selected features lying outside the bounding box, divided by the total number of selected features in the image (averaged over all positive test images). Essentially FPR measures the (inverse) signal-to-noise ratio output by our method, *i.e.* the proportion of irrelevant features it delivers (the lower the better). We compare our method against a baseline, where the confidence for a feature is computed as follows. For each visual word in the codebook we count how many times it appears inside the bounding-box annotations of the training data. This way a visual word, which appears often on the annotated training objects is weighted higher.

On a test image, we match features to the codebook and define BBHR by summing up the weighted matches for each feature. That is, instead of using configurations of features like our system does, the baseline consists of weighted single feature matches – essentially a bag-of-words scheme. This allows to compare our method to the default input to an object recognition system.

Figure 3.13 shows FPR on the y-axis and BBHR on the x-axis, for $k = 5$ and for each dataset. The error bars show the standard deviation of the FPR at a given BBHR. Curves are generated by varying the selection threshold over the feature confidences. As the plots show, our feature selection method is very precise, in that it consistently delivers a low FPR (always below 20%, but for high BBHR on the Cars Rear dataset, where it grows to a moderate 35%). This is an important characteristic, because it enables later processes to rely on a clean input, composed of a large majority of features on the object. This appears especially valuable when compared to the low signal-to-noise ratio of the initially extracted features (there are typically $500 - 1000$ features in an image, out of which about $10 - 200$ lie on the object). The experiments also reveal the substantial performance improvement over the baseline, which we outperform substantially.

The feature selection ability comes at a low price in terms of missed objects: on three of the datasets our method selected at least 5 features (typically many more,

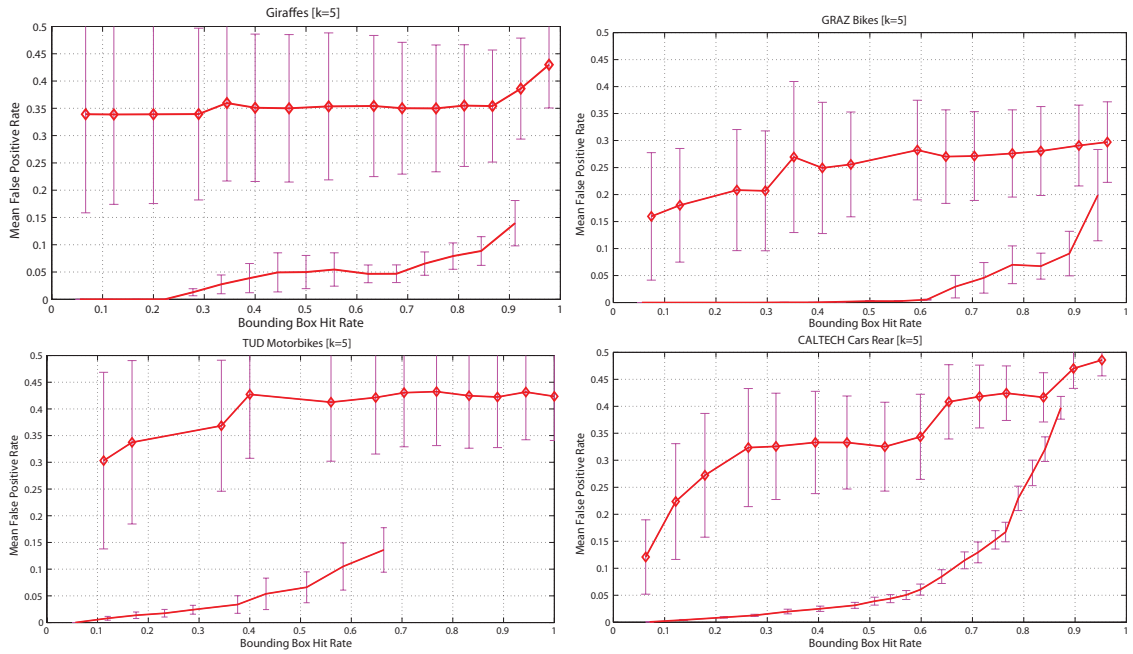


Figure 3.13: *Bounding box hit rates for Giraffes, Bikes, Motorbikes, and Cars Rear Views (lower is better, baseline with diamond marker).*

as in figure 3.12) on about 90% of the object instances. The lower BBHR on the TUD Motorbikes might be due to an excessively high support threshold for mining or a bad visual vocabulary.

The second experiment evaluates our method on the negative test sets (*i.e.* on image without any instance of the object class). The idea is to measure how distinctive the method is: does it select very few features on negative images? This is relevant because the number of features selected on negative images relates to the computational resources the later processing stages will waste on irrelevant data (and to the chances they will get confused and produce wrong results). Figure 3.14 reports the percentage of negative images (y-axis) where at most v features are selected (x-axis). The feature selection threshold is left fixed for each curve, to the one yielding 70%/90% BBHR on the positive dataset (a sensible operating point). As the plots show, at 70% BBHR the method returns extremely few features on the negative images of giraffes and bikes (on 90% of the images it returns less than 3 features). As in the previous experiment, the performance is lower on Motorbikes, but it remains good (in 70% of the images it returns less than 8 features). As expected, at the challenging operating point of 90% BBHR the method returns more features. Nevertheless, it remains distinctive even in this case: 1 in 3 negative images have no selected features, and 70% of the images have less than 10 (remember, we start from $500 - 1000$). The baseline is evaluated in the same manner as for the BBHR plots, and it performs considerably worse than our method.

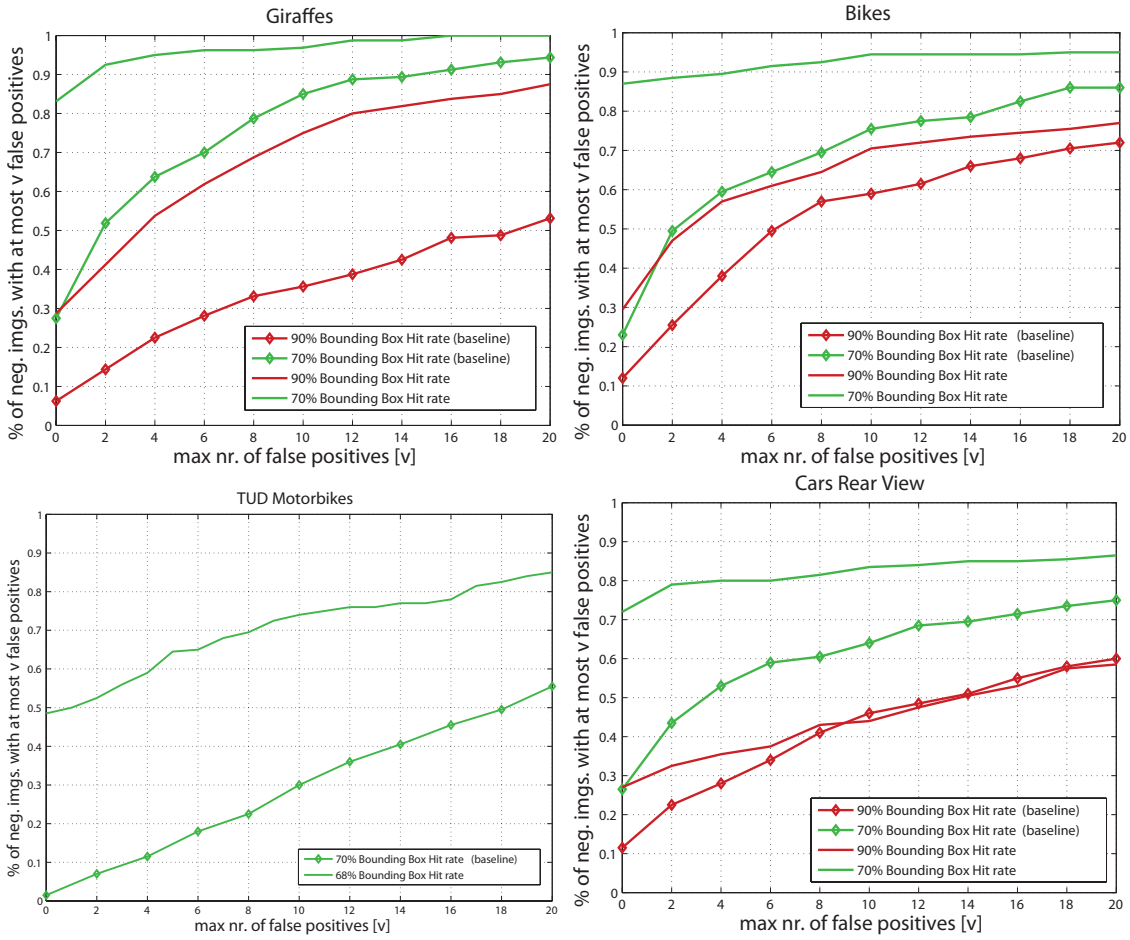


Figure 3.14: *False positives on negative test images for Giraffes, Bikes, Motorbikes, Cars Rear View (higher is better). For the motorbikes we show the experiment for the threshold at 68% BBHR since this is the maximum we reached.*

Data	T	$supp_{min}/conf_{min}$	Q	t CPU
Giraffes	26054	0.20% / 100%	9	2.58 s
Bikes	42390	0.25% / 95%	9	0.91 s
Motorbikes	29001	0.28% / 100%	9	0.90 s
Cars Rear	74296	0.1% / 90%	9	53.02 s

Table 3.2: *Statistics for the mining experiments. Columns: Number of Transactions T, minimal support and confidence thresholds, number of tiles Q, CPU time (in seconds).*

Computation times

The CPU-time measurements are given in Table 3.2. The time is measured for the frequent itemset mining stage including rule creation, but after feature extraction and neighborhoods construction. This because the required processing can be done offline and the required time scales linearly with the number of images. For the mining we use an implementation of the APriori algorithm from [Borgelt, 2003]. All experiments were done on a 3 GHz Intel Pentium 4 with $1GB$ RAM. These measurements demonstrate the scalability of our mining approach, where the most characteristic feature configurations can be extracted from tens of thousands of candidates in a matter of seconds. The mined configurations might be used readily within other frameworks. Table 3.2 also summarizes the mining parameters used for each dataset.

In summary, our experimental evaluation demonstrates that the class-specific confidence measure acts as a good feature selector. Hence, our technique offers a valuable intermediate layer between feature extraction and object detection or other higher-level processes.

3.4 From Frequent Configurations to Objects

In the preceding section we proposed a method, which mines class-specific frequent feature configurations. In this section we try to feed these configurations into an existing object class recognition framework, which builds on (single) instances of local features. More specifically, we combine our method with the Implicit Shape Model (ISM) of [Leibe and Schiele, 2003; Leibe *et al.*, 2005].

This combined recognition process can be summarized as follows: The frequent feature configurations mined in the previous sections represent semantic and discriminant fragments of objects. By matching these fragments to the same kind of semi-local neighborhoods in candidate images, we can estimate the probability for the presence of the object class. To that end, we first collect activations of the rules on the training data. In a second step, following the ISM framework, the activations are used to vote in a Hough voting space for possible object locations, followed by a mean-shift search for maxima in the voting space. The following subsections describe these steps in more detail.

3.4.1 Review of the ISM Approach

In this section we summarize the main concepts of the Implicit Shape Model (ISM). For that purpose we follow closely the description in [Leibe *et al.*, 2008] and adapt notation to our framework where necessary.

The ISM forms the core of a coupled object categorization and figure-ground segmentation method proposed by [Leibe and Schiele, 2003; Leibe *et al.*, 2005]. It is a learned representation for object shape that can combine the information observed on different training examples for recognition using a probabilistic extension of the Generalized Hough Transform [Ballard, 1981; Lowe, 2004]. The $ISM(V) = (V, P_V)$ consists of a class-specific visual vocabulary V and a learnt spatial probability distribution P_V, which specifies where each visual word $c_k \in V$ may be found on the object.

After a visual vocabulary has been created, the model training procedure proceeds with learning P_V. This is done by collecting all occurrences of the visual words $c_k \in V$ and keeping their locations ℓ relative to the object center (o_x, o_y):

$$\ell_x^{(k)} = (c_x^{(k)} - o_x) \tag{3.6}$$

$$\ell_y^{(k)} = (c_y^{(k)} - o_y) \tag{3.7}$$

$$\ell_s^{(k)} = c_s^{(k)} \tag{3.8}$$

For each visual word c_k a list $\mathcal{L}(k)$ of its occurrences is kept, *i.e.* P_V is expressed in $\mathcal{L}(k)$.

Recognition is done using the learnt $ISM(V)$ in a Generalized Hough Transform. To test a novel image for the presence of the learned object class, we first extract its features and match them to the visual vocabulary V. Each matched feature then casts votes for possible position of the object center according to the learned spatial distribution P_V. Consistent hypotheses are then searched as local maxima in the voting space.

The Hough voting space is 3-dimensional with the dimensions $x, y, scale$. Coordinates for votes are thus given as follows

$$x_{vote} = x_{img} - \ell_x^{(k)} * (s_{img}/\ell_s^{(k)}) \tag{3.9}$$
$$y_{vote} = y_{img} - \ell_y^{(k)} * (s_{img}/\ell_s^{(k)}) \tag{3.10}$$
$$s_{vote} = s_{img}/\ell_s^{(k)} \tag{3.11}$$

where $(x_{img}, y_{img}, s_{img})$ is the location of an image feature that could be matched to a visual word and $(x_{occ}, y_{occ}, s_{occ})$ is the k^{th} item from the list of occurrences $\mathcal{L}(k)$ for that visual word on the training data.

Finding the potential locations of an object consists now of finding maxima in the Hough voting space. [Leibe *et al.*, 2008] propose to use a Mean-Shift search procedure to identify maxima robustly and efficiently.

3.4.2 Recognition with Rule Activations

We now formulate object class detection using a combination of mined frequent feature configurations and the ISM approach summarized above. The main difference is, that votes do not originate at all feature locations, but only at locations of matched frequent configurations. Thus, the slightly adapted derivation is now as follows.

Suppose we have a set of q annotations for objects on training data. For each annotation, the mined frequent configurations $\mathcal{C}$ are matched to the neighborhoods $\mathcal{P}$ within the annotation area according to Equation (3.4). For each matched configuration we record the relative position ℓ of the object center (o_x, o_y):

$$\ell_x^{(k)} = (n_x^{(k)} - o_x) \tag{3.12}$$
$$\ell_y^{(k)} = (n_y^{(k)} - o_y) \tag{3.13}$$
$$\ell_s^{(k)} = n_s^{(k)} \tag{3.14}$$

where $(n_x^{(k)}, n_y^{(k)}, n_s^{(k)})$ refers to the position of the central region of the kth neighborhood matched with the rule. For each rule $(\mathcal{P} \rightarrow object)$ we keep a list of all the activations $\mathcal{L}(\mathcal{C})$ with relative positions of the object center.

With the collected evidence we can now detect and locate object candidates in previously unseen images. As proposed in [Leibe *et al.*, 2005] we collect votes for the object center location in a three dimensional Hough voting space. That is, we use our discriminate frequent configurations to detect objects in a generative recognition framework. In each candidate image we match again the mined frequent configurations $\mathcal{C}$ to the neighborhoods $\mathcal{P}$ as in equation (3.4). For each match $m(\mathcal{C}, \mathcal{P})$ we vote in a scale invariant manner with the activations $\ell^{(k)}$ from the list $\mathcal{L}(\mathcal{C})$.

$$vote_x = m_x - \ell_x^{(k)} * (m_s / \ell_s^{(k)}) \tag{3.15}$$

$$vote_y = m_y - \ell_y^{(k)} * (m_s / \ell_s^{(k)}) \tag{3.16}$$

$$vote_s = (m_s / \ell_s^{(k)}) \tag{3.17}$$

where $\{m_x, m_y, m_s\}$ stands for the (x, y, s)-location of the neighborhood $\mathcal{P}$ from the match $m(\mathcal{C}, \mathcal{P})$.

Similar to the derivation in [Leibe *et al.*, 2008], this Hough voting procedure can also be expressed in a probabilistic framework. Given a set of neighborhoods $\mathcal{P}$ we want to determine the probability of the existence of object o_n at location x. By matching $\mathcal{P}$ to the frequent configurations $\mathcal{C}$ the voting can be formulated as follows:

$$p(o_n, x|\mathcal{P}) = \sum_i p(o_n, x|\mathcal{C}_i, \mathcal{P}) p(\mathcal{C}_i|\mathcal{P}) \tag{3.18}$$

$$\approx \sum_i p(o_n, x|\mathcal{C}_i) p(\mathcal{C}_i|\mathcal{P}) \tag{3.19}$$

The simplification of the first term in equation (3.19) is justified, since after mining only the frequent configurations influence the estimated location of the object o_n. It follows further

$$p(o_n, x|\mathcal{P}) = \sum_i p(x|o_n, \mathcal{C}_i) p(o_n|\mathcal{C}_i) p(\mathcal{C}_i|\mathcal{P}) \tag{3.20}$$

$$= \sum_i \frac{|\mathcal{L}(\mathcal{C}_i \mid d(vote_i, x) \leq \varepsilon)|}{|\mathcal{L}(\mathcal{C}_i)|} * conf(\mathcal{C}_i \rightarrow o_n) * m \tag{3.21}$$

The first term in equation (3.20) is the Hough vote for position x given for object o_n given the rule $(\mathcal{C} \rightarrow o_n)$. The second term is the confidence that the presence of the object o_n can be inferred from the configuration $(\mathcal{C}_i)$, and the final term is the probability that the rule is active. The individual terms can be directly replaced by the terms shown in equation (3.21), where $|\mathcal{L}(\mathcal{C}_i)|$ is the length of the activation list $\mathcal{L}(\mathcal{C}_i)$ and $|\mathcal{L}(\mathcal{C}_i \mid d(vote_i, x) \leq \varepsilon)|$ is the length of the partial list voting for position x. The last term m is the match indicator from equation (3.4).

After filling the voting space, maxima in the space are found with a mean-shift search. Each maximum generates a hypothesis $h(x, y, s)$ with a score derived from

the value of the respective maximum. Overlapping hypotheses are treated with an overlap filter, where from two overlapping hypotheses the one with the higher score survives.

3.4.3 Experiments and Results

In this section we validate our approach quantitatively by a series of experiments. A discussion of the mining parameters is followed by a recognition evaluation in an object class detection task. The experiments were conducted on two datasets:

TUD Motorbikes. (See previous section for description). As background data for this class a randomly selected subset of the CALTECH-256 [Griffin *et al.*, 2007] background class was used. The detections are counted as correct if their bounding box matches the ground-truth annotation (with intersection-over-union > 0.5) and extra hypotheses are counted as false positives.

UIUC Cars. The UIUC single-scale test set [Agarwal *et al.*, 2004] consists of 170 images containing 200 side views of cars of approximately the same size. Challenges include partially occluded cars, instances that have low contrast with the background, and images with highly textured backgrounds.

Tiling Parameters

To investigate the effect of the number of tiles T for the creation of the neighborhoods $\mathcal{N}$, we ran experiments with different T on the TUD motorbikes dataset [Various, 2005] ranging from a 3x3 to a 6x6 tiling. The results are shown in Figure 3.15. (Curces are generated by varying through the confidence values for detactions, obtained by aggregating the Hough votes). The neighborhood size was set to $S = 5$ times the size of the central regions R_c. The mining parameters were held constant at $s_{min} = 0.5\%$ support and $c_{min} = 90\%$ confidence. The best results are obtained for 9 and 25 tiles. At first sight it is unexpected, that 9 tiles perform better than the versions with 25 and 16 tiles. However, since the neighborhood size and the mining parameters are kept constant, the version with 9 (larger) tiles generates more rules. Having more evidence leads usually to better performance. Increasing the number of tiles to 36 leads to higher precision in the low recall area, but is punished with less recall overall. It is somewhat surprising, that the 15 tiles version performs better than the 16 tiles configuration. It seems that the mining is easily influenced in an uncontrolled way by changes in the underlying data, at least on this particular dataset.

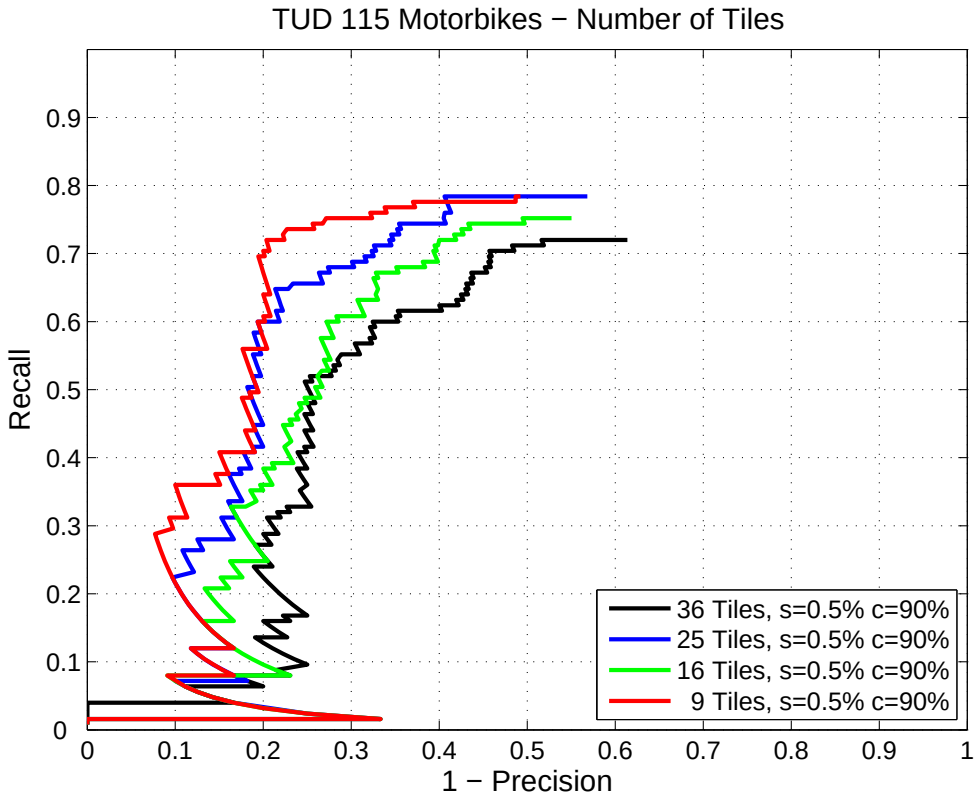


***Figure 3.15**: Performance per # tiles on TUD Motorbikes.*

Mining Parameters

The following set of experiments discusses the effects of the mining parameters. The first experiment deals with the minimal confidence c. Figure 3.16(left) shows recognition results for different minimal confidence thresholds. Using confidence values around 80% seems to be the best trade-off between robustness of the rules against background data and low recall values caused by rules that are too specific. Going to the extreme of using only 100% confidence rules decreases recall dramatically while giving nearly no improvement in precision. Using very low confidence of 30% increases the danger of including false hypotheses with high weight, as can be seen from the dent of the corresponding curve at low recall. The next experiment looks into the minimal support threshold and is shown in the middle plot of Figure 3.17. Again, we measure the overall recognition rates, this time at different support values. The confidence is held constant at 80%. Using only the extremely frequent configurations with more than 2% support for detection leads to bad performance. This can be explained by the properties of the TUD and CALTECH motorbikes databases: The CALTECH images used for mining the rules in this specific example contain for

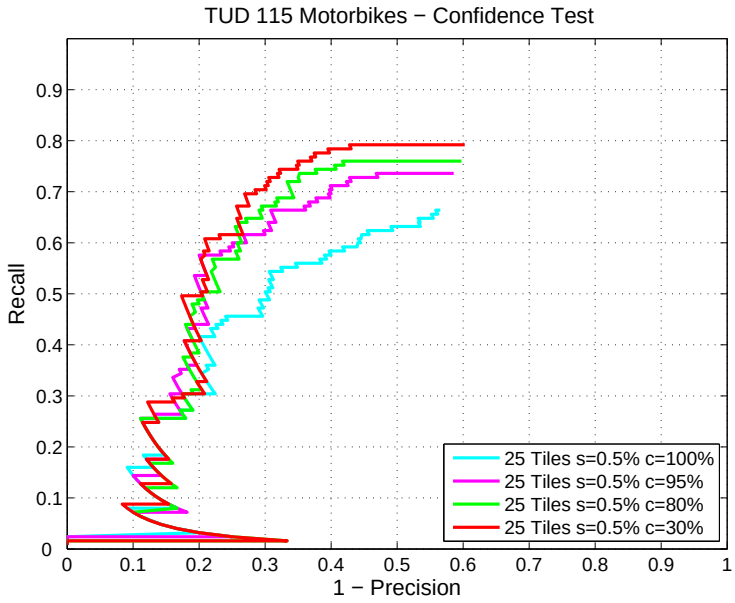


Figure 3.16: *Recognition performance for minimal confidence values*

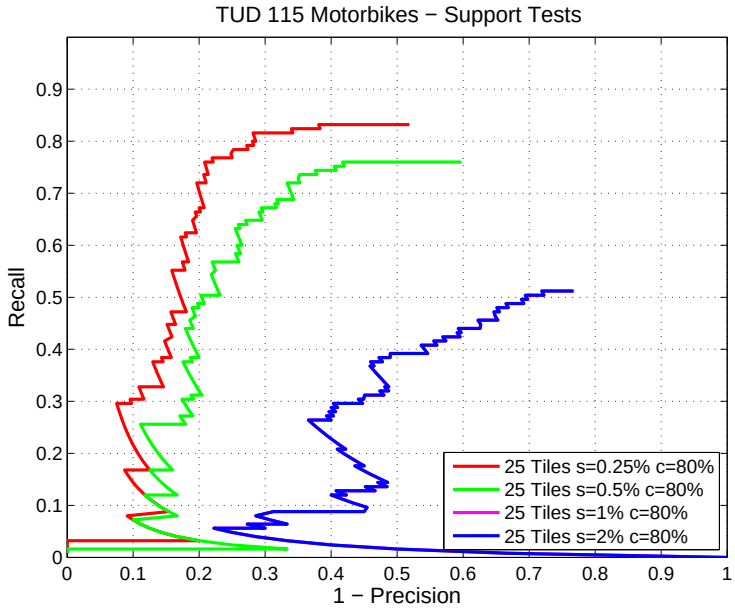


Figure 3.17: *Recognition performance for minimal support values*

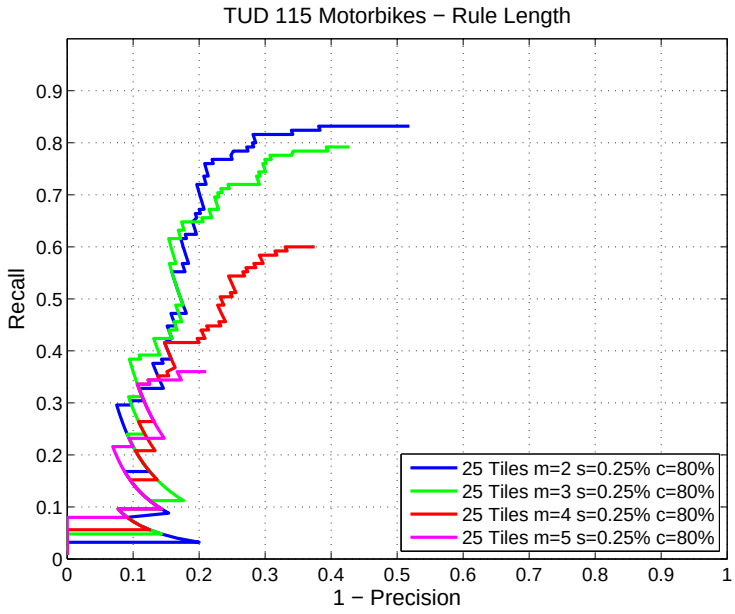


***Figure 3.18**: Recognition performance for rulelengths*

the most part very clean images of motorbikes in front of white background, such that even small evidence has mostly positive influence on the recognition rate.

A last experiment was conducted to measure the influence of the length of a rule. That is, how many features are part of a mined spatial configuration. The recognition for different rule lengths are shown in the rightmost plot of Figure 3.18, where m denotes the minimal rule-length. Using a minimal rule length of $m = 3$ seems to be the best trade-off between specificity and generality of the frequent configurations. Not surprisingly, using only feature configurations with $m = 5$ or more elements leads to better precision at the cost of recall.

Mining Performance

This section discusses some of the properties of the frequent itemset mining method. For the mining we use an implementation of the APriori algorithm from [Borgelt, 2003]. All experiments were done on a 3GHz Intel Pentium 4. The CPU-time measurements are given for some examples in table 3.3. The time is measured for the frequent itemset mining step including rule creation, *i.e.* without feature extraction and neighborhoods already created. This is usually the case for large databases, since the required processing can be done offline. The measurements demonstrate the scalability of the mining approach, where the most characteristic configurations can be extracted from hundreds of thousands of candidates in a couple of seconds. This clearly shows the benefits of our approach. Most of the current approaches

Data	# Transact.	CPU Time	$supp_{min}/conf_{min}$
TUD	20771	2.46 s	0.25% / 80%
UIUC	113156	3.49 s	0.25% / 95%

***Table 3.3**: Mining statistics for the experiments. Columns: Number of Transactions T, CPU Time and minimal support and confidence thresholds.*

which rely on local features for object detection are built on single instances of these features, or become quickly too complex to handle efficiently. With mining approach we avoid such limitations. The mined configurations might be used readily within other frameworks.

Object Detection

To demonstrate the performance of our method, we evaluate it on two visual category detection tasks:

The plot Figure 3.20 (right) shows the performance measurement results for the TUD motorbikes. In [Fritz *et al.*, 2005] 81% EER are reported on this set. With our approach we achieve an EER of 77%, which is comparable. Since the approach in [Fritz *et al.*, 2005] uses a verification stage with a Minimum Description Length (MDL) filter and an optional additional SVM layer. Our simple bounding-box-overlap filter could be replaced by this verification stage. This would probably increase performance to the same level, since many of the false positives in our system are caused by overlapping objects, which are falsely removed by the bounding-box overlap filter. Figure 3.21 shows a few examples of detections on the motorbike set. Correct detections are shown in yellow, false detections in red. Detections are robust to clutter and cover examples in a variety of contexts. The wrong detection can be explained by the covered rear wheel, which makes the precise estimation of the object center difficult in this example. For the evaluation on the UIUC dataset we followed the protocol of [Agarwal *et al.*, 2004] and used the evaluation software provided with the dataset. A codebook was created on the provided training images. Frequent configurations were mined on the same set, using the provided background set to find discriminative rules and configurations as discussed in section 3.3.1. The results on this testset are shown in Figure 3.19. The Equal Error Rate (EER) on the UIUC single-scale reaches 91%, which is within the state of the art.

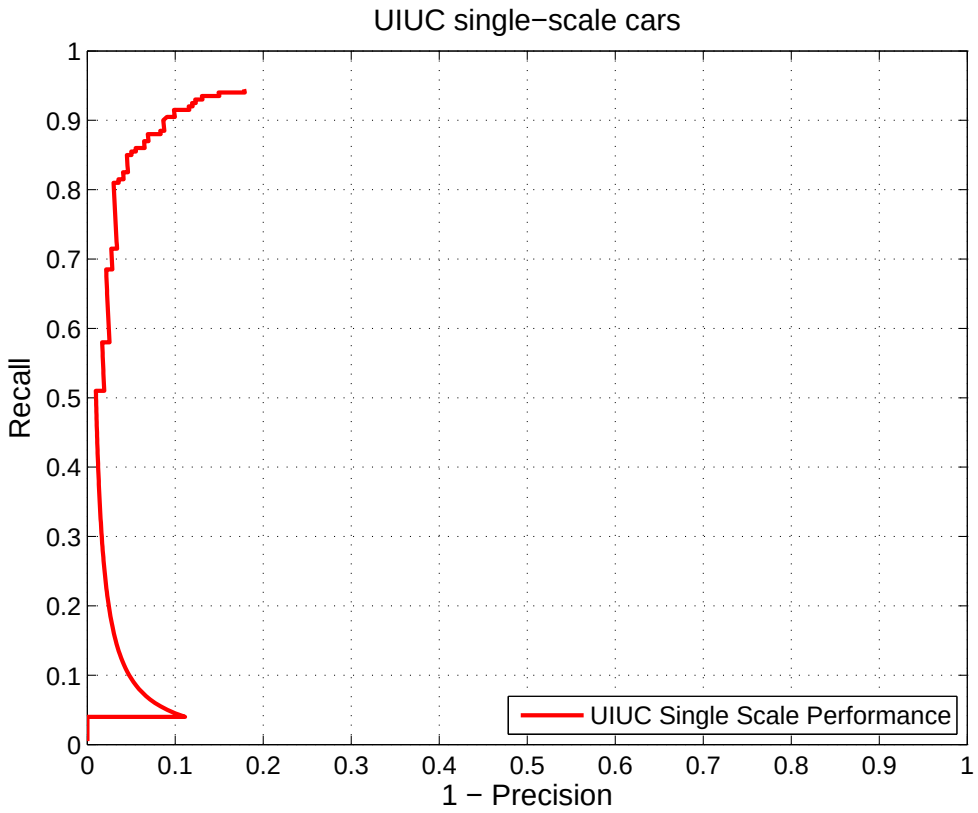


Figure 3.19: *Performance on UIUC. EER (91%)*

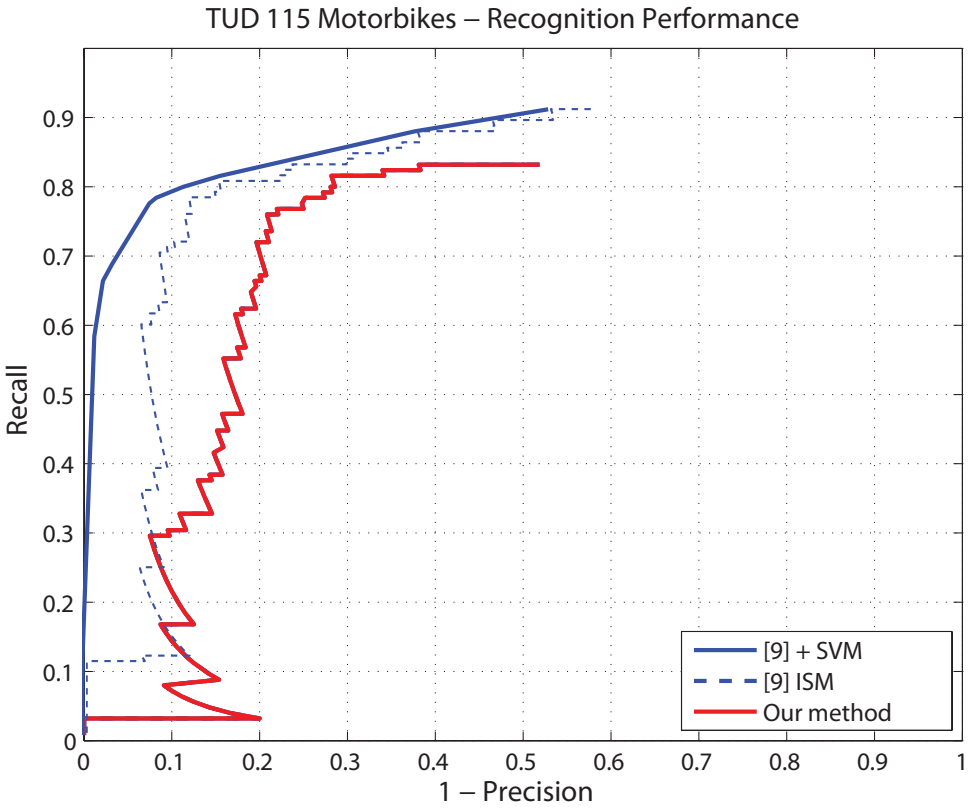


Figure 3.20: *Performance on TUD Motorbikes. (EER 77%)*

Figure 3.21: *Examples of three correct and one false detection on the TUD motorbikes set.*

3.5 Graph Mining as an Alternative to Itemsets

In the previous sections we built on frequent itemset mining to detect frequent combinations of visual words and extended the method to encode spatial configurations in a semi-local grid. An alternative way to encode spatial relations between local features is by the means of a graph. The nodes consist of the images' features labeled with their visual word ids, and the edges describe the spatial relation between features in the image plane. The idea is then to apply mining algorithms, which operate directly on the graph to identify frequently occurring sub-graphs, which are correlated with an object or object class. Some of the most popular algorithms in this field are described in Section 2.5.

The collected the evidence in form of frequent sub-graphs can be seen as weak classifiers to decide on the presence or absence of an instance of the object class the graphs were mined for. Thus it is quite natural a next step to add a boosting method on top of these simple classifiers to learn a stronger classifier for the given object class.

These two steps, mining of frequent sub-graphs and their combination into a classifier using boosting, are described in the following subsections in detail. Please note that this chapter is a summary of joint work with my former Master student Sarah Gugl. The interested reader can find further details in her report [Gugl, 2007].

3.5.1 Mining of Frequent Feature Graphs

As in the previous sections about itemset mining, the image content is first encoded using local features which are quantized into a visual vocabulary. Here we use SIFT features [Lowe, 2004].

The visual words and their relative positions are now encoded in a graph for each image as follows. Node labels are simply the visual word id. Edge labels encode the relative positions of the features in a canonical way. To that end we use the simple encoding schemes as shown in Tables 3.4 and 3.5, respectively. The first labeling method only encodes the relative position of features, the second method also includes their relative scales. In both variants canonical labeling is achieved by following the convention that relations are expressed relative to the node with the smaller label (*i.e.* smaller visual word id). Note that we don't encode distances (distances between nodes or scale difference), but only smaller/larger relationships.
Using these labeling conventions, we have several options for building a graph per image:

- Complete Graph

edge label	relative position of the end-vertices
0	$x_i > x_j \quad \wedge \quad y_i > y_j$
1	$x_i > x_j \quad \wedge \quad y_i \leq y_j$
2	$x_i \leq x_j \quad \wedge \quad y_i > y_j$
3	$x_i \leq x_j \quad \wedge \quad y_i \leq y_j$

***Table 3.4**: Edge labels for the edge connecting vertices i and j $(i < j)$.*

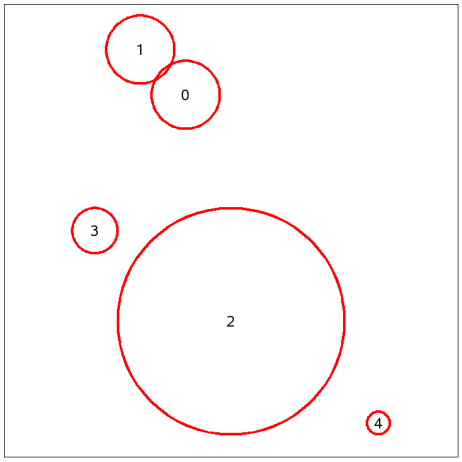
edge label	relative position of the end-vertices
0	$x_i > x_j \quad \wedge \quad y_i > y_j \quad \wedge \quad scale_i > scale_j$
1	$x_i > x_j \quad \wedge \quad y_i > y_j \quad \wedge \quad scale_i \leq scale_j$
2	$x_i > x_j \quad \wedge \quad y_i \leq y_j \quad \wedge \quad scale_i > scale_j$
3	$x_i > x_j \quad \wedge \quad y_i \leq y_j \quad \wedge \quad scale_i \leq scale_j$
4	$x_i \leq x_j \quad \wedge \quad y_i > y_j \quad \wedge \quad scale_i > scale_j$
5	$x_i \leq x_j \quad \wedge \quad y_i > y_j \quad \wedge \quad scale_i \leq scale_j$
6	$x_i \leq x_j \quad \wedge \quad y_i \leq y_j \quad \wedge \quad scale_i > scale_j$
7	$x_i \leq x_j \quad \wedge \quad y_i \leq y_j \quad \wedge \quad scale_i \leq scale_j$

***Table 3.5**: Edge labels for the edges connecting vertices i and j $(i < j)$*

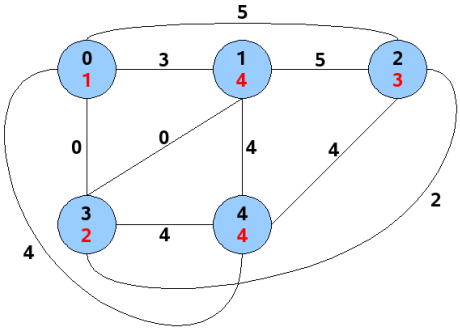
- k-Nearest Neighbors of selected features (kNN)
- Neighborhood defined by area around selected features

The 2nd and 3rd option are considered due to scalability reasons: mining fully connected graphs is computationally more complex. Connecting all n vertices by an edge yields $\frac{n(n-1)}{2}$ edges. Assuming a typical image contains about 500 features would result in a graph with $124'750$ edges. A toy example of such a graph is shown in Figure 3.22(b). Using the kNN approach instead, results not in a single large graph per image, but in multiple smaller graphs, *i.e.* for n images with $m_1, m_2, \ldots, m_n$ features $\sum_1^n m_n$ graphs will be created in total. So the the mining problem shifts from mining few large graphs to mining many smaller graphs. Expressing a local neighborhood by the selection of the k nearest neighbors around each feature is inspired by [Sivic and Zisserman, 2004]. The resulting graph for a toy example is shown in Figure 3.22(c). Following our approach from the previous sections about itemset mining, a neighborhood can also by obtained by defining a scale-invariant area around a feature. The difference to the previous sections is that we don't encode feature's locations within the area using a tiled grid, but use the graph to encode the relative positions of features within the neighborhood. The resulting graph for a toy example is shown in Figure 3.22(d).

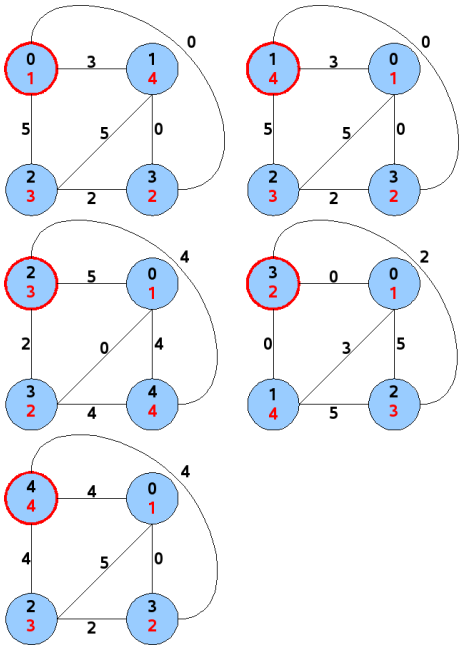
Having the images encoded as graphs, the next step consists of mining frequent subgraphs. As outlined in Section 2.5 we can choose among several methods to solve this task. From those we selected FSG, gSpan/CloseGraph and MoSS/MoFa for closer inspection. (SUBDUE did not qualify because it is an approximate method, AGM



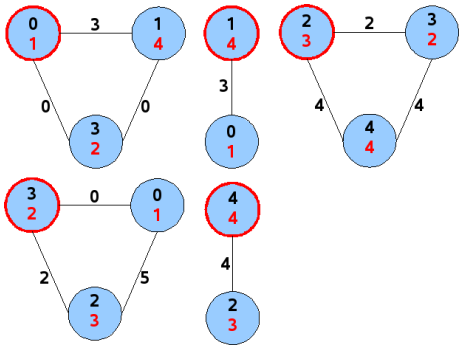
(a) Toy example image



(b) Fully connected graph for the example image (edge labeling method A)



(c) k-nearest neighbor graphs for $k = 3$



(d) Neighborhood area graphs

***Figure 3.22**: Example image and resulting graph variants.*

	FSG	gSpan/ CloseGraph	MoSS/MoFa
Search	breadth-first	depth-first	depth-first
TID-Lists	yes	yes	yes
Runtime ($s = 5\%$)	a few minutes	a few minutes	hours
largest subgraph for $s = 10\%$	2 edge	1 edge	1 edge
♯ graphs	*supp* *num* 20% 0 15% 2 10% 32 7.5% 124 6% 274 5.5% 447 5% 447	*supp* *num* 20% 0 15% 2 10% 22 7.5% 103 6% 247 5.5% 405 5% 700	*supp* *num* 20% 0 15% 2 10% 22 7.5% 103 6% 247 5.5% 405 5% –
largest graph	supp ♯edges 20% 0 15% 1 10% 2 7.5% 3 6% 10 5.5% 19 5% 19	supp ♯edges 20% 0 15% 1 10% 1 7.5% 3 6% 6 5.5% 10 5% 19	supp ♯edges 20% 0 15% 1 10% 1 7.5% 3 6% 6 5.5% 10 5% –

***Table 3.6**: Comparison of FSG, CloseGraph and MoSS/MoFa based on implementations downloaded from the internet.*

since it does not support edge labeling.) For each of the algorithms we retrieved an implementation from the respective authors' websites and run tests on a sample graph. This graph set was constructed from the Caltech-4 motorbikes set [Fergus *et al.*, 2003] by extracting SIFT features and clustering them into 2788 visual words. Graphs were constructed as proposed above, with visual word ids as node labels and encoding relative positions of the nodes as shown in Table 3.5. From Table 3.6 it can be seen that FSG is the fastest algorithm, followed by gSpan/CloseGraph. The slowest algorithm is MoSS/MoFa which could not finish the computation within reasonable time for a support of 5%. The numbers of retrieved subgraphs differ for the same support value. This could be due to differences in the ways of computing the support values, for example the usage of different rounding methods. While FSG and gSpan seem to be quite similar, we chose gSpan for our further experiments because it was shown to outperform FSG in [Yan and Han, 2002].

Figure 3.23 shows the result of a gSpan run on a real dataset. The dataset is the caltech-4 motorbikes set [Fergus *et al.*, 2003]. This dataset consists of 153 images and gSpan was run with a support of 5%. In figure 3.23 one subgraph is shown, consisting of seven vertices and ten edges and occurring in eight images. As can be seen from the figure, the subgraph has the same semantical meaning in all images, i.e. its location is roughly the same on all images — on all eight images out of the seven activated features, three lie on the back wheel, two on the front wheel, one between the back wheel and the seat and the last one between the front wheel and the fender. These figures show, that just like frequent itemsets, frequent subgraphs have the pleasant property to be easily interpretable.

3.5.2 Classification using Boosting

Having mined frequent subgraphs we want to investigate if they are a suitable choice for object class recognition and detection. Instead of combining them with an existing object recognition framework as we did with the frequent itemsets, here we treat the graphs as simple classifier and combine them into a stronger classifier using boosting.

That is, we want to find a classification rule that constructs a graph G_I for any given image I and then decides from G_I and subgraphs previously mined on a training set if the image I contains an instance of the trained object class. Thus, the input to our classification rule is a set of subgraphs $\{s_1, s_2, \ldots, s_n\} = \mathcal{S}^n$ and an image graph $G_I \in \mathcal{G}$, and the output is a class label $y \in \mathcal{Y}$. In other words the goal is to find a classification rule

$$h : \mathcal{S}^n \times \mathcal{G} \rightarrow \mathcal{Y} \tag{3.22}$$

A simple classifier, which adheres to this rule can be constructed by counting the occurrences of the mined subgraphs $s_i \in S$ in the graph G_I and base the classification decision on a threshold t for the count value:

$$h(S, G_I) = \begin{cases} 0, & numEmbeddings < t \\ 1, & numEmbeddings \geq t\,, \end{cases} \tag{3.23}$$

where t is the predetermined threshold value and $numEmbeddings$ is the number of subgraph occurrences in the graph G_I. This number is computed by computing a subgraph-graph-isomorphism for all subgraphs $s_i \in S$ and G_I, which for all subgraphs s_i gives a number $n_i = SubgraphGraphIso\,(s_i, G_I)$ that counts the embeddings of s_i in G_I. $numEmbeddings$ then is determined by summing up all these numbers:

$$numEmbeddings = \sum_i SubgraphGraphIso\,(s_i, G_I)\ .$$

To boost the performance of our simple classifier we use AdaBoost [Freund and Schapire, 1997]. Our simple classifier (equation 3.23) is now extended with a weight

Figure 3.23: *In all these images a frequent subgraph consisting of seven vertices and ten edges is shown. As can be seen, the subgraph has the same semantical meaning in all images*

w_i for each subgraph. Taking the weighted sum of the sub-graph counts leads to the following confidence value $conf_{G_I}$ for the graph G_I of the image I:

$$conf_{G_I} = \sum_i w_i * n_i \,.$$

In this manner for each graph a confidence value can be obtained. The weak classifier now chooses a threshold, that minimizes the error rate:

$$T = arg\left(\min_t\left(\sum_i \mathbb{I}_{h_t(x_i)\neq y_i}\right)\right) \,,$$

where $h_t(x_i)$ is the classifier obtained by choosing t as threshold T. The final classifier then returned to AdaBoost is

$$h_T(x) = \begin{cases} 1 & \text{if } conf_{G_x} \geq T, \\ -1 & \text{otherwise.} \end{cases}$$

From the weights W_I obtained from the re-weighting procedure of AdaBoost the subgraph weights w_i can be computed by determining a positive and negative weight for each subgraph. The positive weight is computed from all images belonging to the class (positive examples) by first determining for each positive example P how often a given subgraph occurs in the graph G_P of this example (giving a number of occurrences n_P) and then multiplying this number by the weight W_P of P and finally summing them up:

$$posWeight_i = \sum_P n_P * W_P \,.$$

The negative weights are obtained similarly, by looking at the images N not belonging to the class (negative examples):

$$negWeight_i = \sum_N n_N * W_N \,.$$

The final weight w_i of the subgraph S_i then can be obtained by computing

$$w_i = \frac{posWeight_i}{negWeight_i + 1}$$

and finally normalizing all weights

$$w_i = \frac{w_i}{\sum_j w_j} \,.$$

	numAssign			
support	1	2	5	10
15%	0.7843	0.7320	0.6013	0.5752
10%	0.3006	0.2353	0.1308	0.0915
7.5%	0.1765	0.1308	0.0327	0.0261
6%	0.1503	0.0850	0.0327	0.0196
5.5%	0.1373	0.0784	0.0261	0.0196
5%	0.1307	0.0654	0.0196	0.0131

Table 3.7*: Error rates obtained by using soft assignments to build the image graphs with* $numAssign = 2$, 5 *and* 10 *and the simple classification rule.*

Matching Graphs

When classifying a previously unseen test image, the first step consists of determining the occurrences of the mined frequent subgraphs in the test image. This requires matching the mined sub-graphs to the image graph(s) using a subgraph-graph isomorphism. Two extensions to a standard isomorphism turned out to be crucial for performance: soft-matching and edge-length filtering.

The effect of the soft-matching feature (*i.e.* assigning feature vectors to several visual word centroids instead of only the single closest one) can be observed from Table 3.7. It shows the error (false negatives) when applying the classifier to the set it was trained on (here for the TUD motorbikes positive training set), depending on the number of visual words features were assigned to. Clearly, assigning to more than one centroid helps to increase recognition rates, while the benefit diminishes when increasing the number of visual words assigning to. In summary, introducing soft-matching helps us reduce false negatives that occurred due to the quantization of the feature space in visual words.

Another observation concerned false positive graph matches, mostly on background data. Investigation of a few sample cases revealed, that many of the false positives appeared due to strongly varying relative edge lengths within the graphs between training and test-data. This is further illustrated by the histogram plots in Figure 3.24. These plots show the distribution of edge lengths for several selected node pairs of frequent subgraphs. The edge length is made scale invariant and canonical by determining it relative to the scale of the node with the smaller label (*i.e.* visual word id). While all the plots show some strong peaks, some of them even form a single peak Gaussian-like distribution. This means that for many sub-graphs the distance between nodes is a relevant feature, which was not included in our graph model, which just contains the relative location of nodes. At the same time it seems to be a feature which is not very stable (not always clear peaks) and also depends on the sometimes very imprecise scale assignment of the feature detector. Thus,

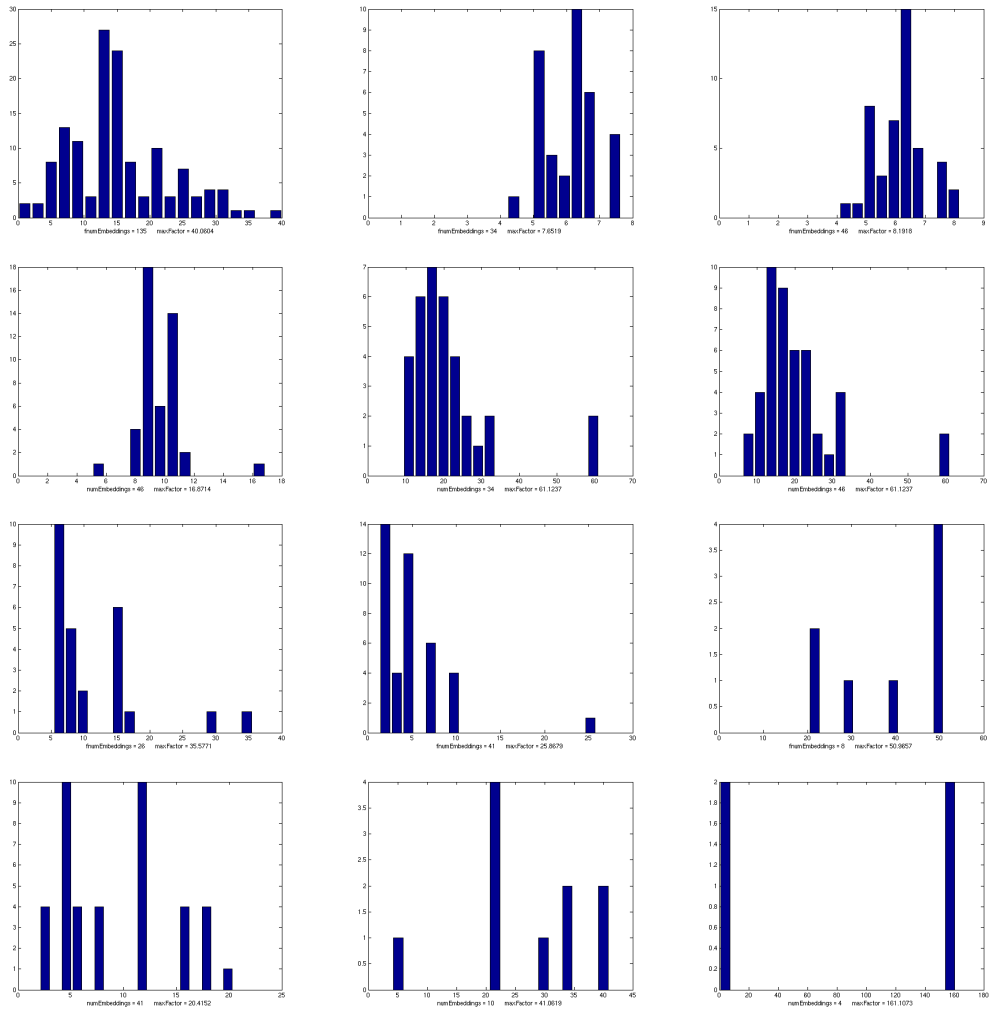


***Figure 3.24**: Distribution of normalized edge-lengths for node pairs*

rather than quantising the edge lengths and encoding it in the graph model, we extended the basic graph isomorphism with an edge-length filter, which rejects edges which do not fall into a valid length-range. The range was determined by selecting peaks higher than $c * \frac{1}{numbins}$ and filtering outside k neighboring bins of the peak. In summary, the combined elements leads to the training and classification procedures summarized in Figures 3.24 and 3.25, respectively.

3.5.3 Experiments and Results

We present experiments and results for several tasks. First, we want to compare the feature selection capabilities of the graph mining algorithm to the ones obtained using itemset mining in Section 3.3. We then continue with classification experiments and some measures on computational performance.

TrainClassifier(*supp*, *numAssign*, *label*, *ratio*, T, $posTrainImg$, $negTrainImg$)
1: Construct the graph set G for posTrainImg;
2: Learn the frequently occurring subgraphs for the support-values *supp*;
3: Determine all occurrences of the frequent subgraphs in the graph set G;
4: Learn the distribution of the edge-lengths;
5: Compute the subgraph-graph isomorphisms on positive and negative training data;
7: Discard the subgraph-occurrences, which do not fall within valid edge length ranges;
8: Train the AdaBoost classifier using the remaining subgraph-graph isomorphisms;

Figure 3.25*: Pseudo-Code representing the overall training procedure. supp is the support-value used while mining frequent subgraphs, numAssign is the soft assignment used while matching to a visual vocabulary, ratio and T are two parameters for the learning of the AdaBoost classifier and $posTrainImg$ and $negTrainImg$ are the images of the positive and negative part of the training set.*

Classify(*supp*, *numAssign*, *label*, $classif$, $testImg$, *intervals*)
1: Compute the graph set G for testImg;
2: Compute the subgraph-graph-isomorphisms;
3: Discard isomorphisms, which do not fall within valid edge length ranges;
4: Classify using the remaining isomorphisms and the given classifier $classif$;

Figure 3.26*: Pseudo-Code representing the final classification procedure. supp is the support-value used for the learning of the frequently occurring subgraphs, numAssign is the soft assignment used while matching to a visual vocabulary, $classif$ is the previously learned AdaBoost classifier, $testImg$ are the images of the testset and intervals are the previously learned intervals for the edge-lengths.*

The evaluation was conducted on the following datasets already introduced in the experiments of the previous chapters:

TUD Motorbikes. The TUD Motorbikes dataset [Various, 2005] consists of 115 images containing 125 motorbikes, which we used as positive test set. The positive training images are the Caltech-4 motorbikes [Fergus *et al.*, 2003] (no bounding-boxes given). As (negative) background training set we randomly picked 180 images from Flickr[4]. We also collected 38 negative test-images.

CALTECH Cars Rear. This dataset features 126 rear-views of cars and 1155 street scenes without cars, used as training set. Moreover, the dataset also provides a test set of 526 images containing cars, as described in [Fergus *et al.*, 2003]. Due to the computational restrictions posed by the runtimes of the graph isomorphisms we

[4]http://www.flickr.com

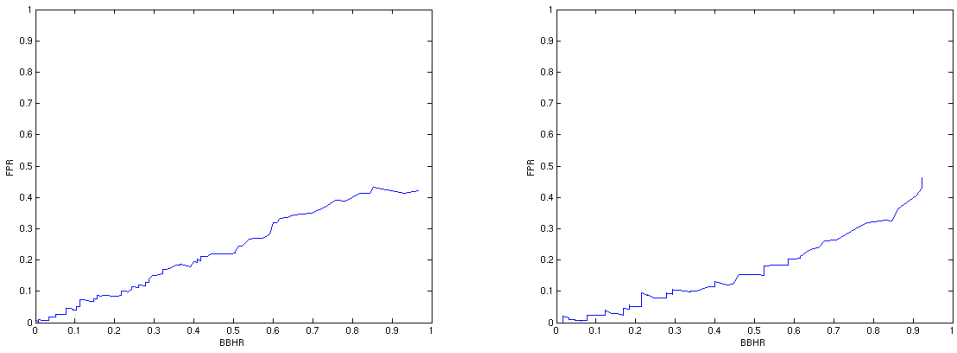


Figure 3.27: *Bounding box hit rate on TUD Motorbikes (left) and Caltech Cars Rear (right) (minimal suport* 11%*, soft assignment parameter* 1 *and* $k = 5$*).*

used only 65 images as testsets. The negative training and test sets consisted again of the 180 and 38 images from Flickr, respectively. For both sets SIFT features were extracted and clustered into a visual vocabulary of 446 visual words using a hierarchical agglomerative clustering algorithm,

Feature Selection

We first show results on feature selection, using the same Bounding Box Hit Ratio (BBHR) measure as in Section 3.3.3. The results are plotted in 3.27 for motorbikes and cars, respectively. Clearly, overall the values are below the ones reported in Section 3.3.3 for itemset mining, in spite a simpler negative testset (*i.e.* one would expect lower FP rates). However, for the TUD motorbikes case, more instances of the object class are detected overall than with itemset mining, which is probably due to soft-matching the visual words. Figures 3.28 and 3.29 show a visualization of feature activation counts. Compared to the results obtained using itemset mining, the features seem to cover larger fraction of the object surface, and there are few discriminant peaks over the false positives occurrences on the background.

Classification

For the classification task we report quantitative results in the form of ROC curves. Experiments were done by varying the parameters through the value ranges shown in Table 3.8. ROC curves were obtained by varying the number of AdaBoost iterations T.The best Equal Error (EER) rates achieved using this method are 70% for the TUD motorbikes and 82% for cars. These results are slightly below state-of-the-art classification results on these datasets.

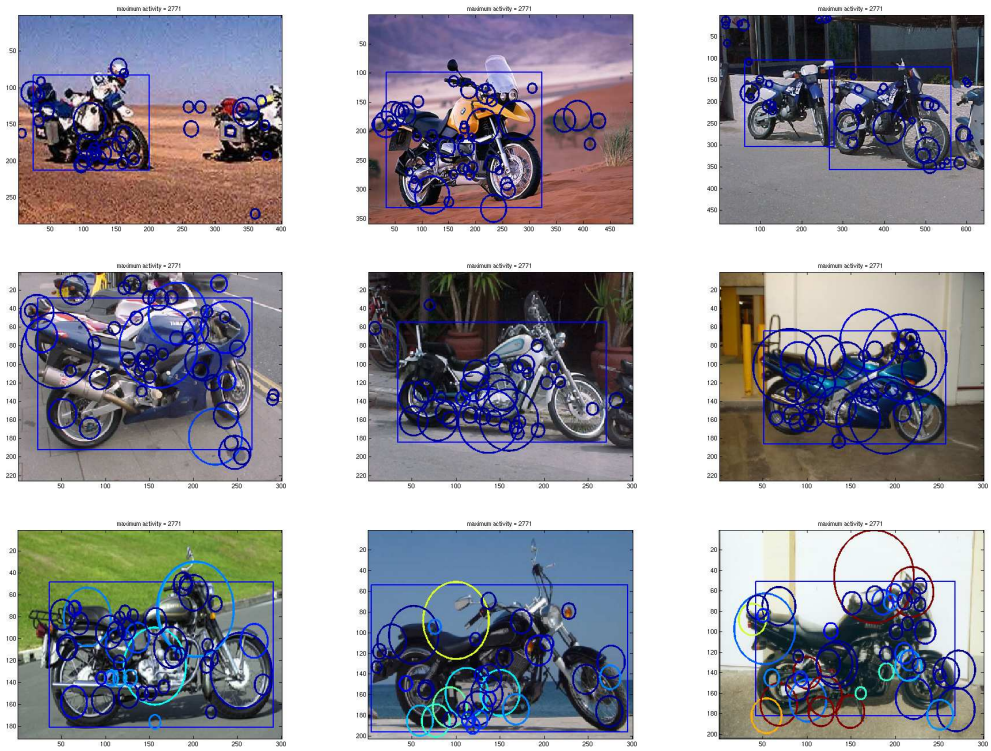


Figure 3.28: *Some examples of the activations of the motorbikes testset. The examples where taken for a support value of* 11%*, the edge-labeling method A and the soft assignment parameter* 1*. The color of the activated features gives the number of activations, where blue means that the feature is not often activated, and red means that the features is activated often.*

Parameter	Values
T	1..50
support	20%, 15%, 12.5%, 12%, 11% and 10%
$soft\_assignment$	1, 2, 5 and 10
ratio	1, 2, 5, 10, 20, 50, 100, 200, 500 and 1000

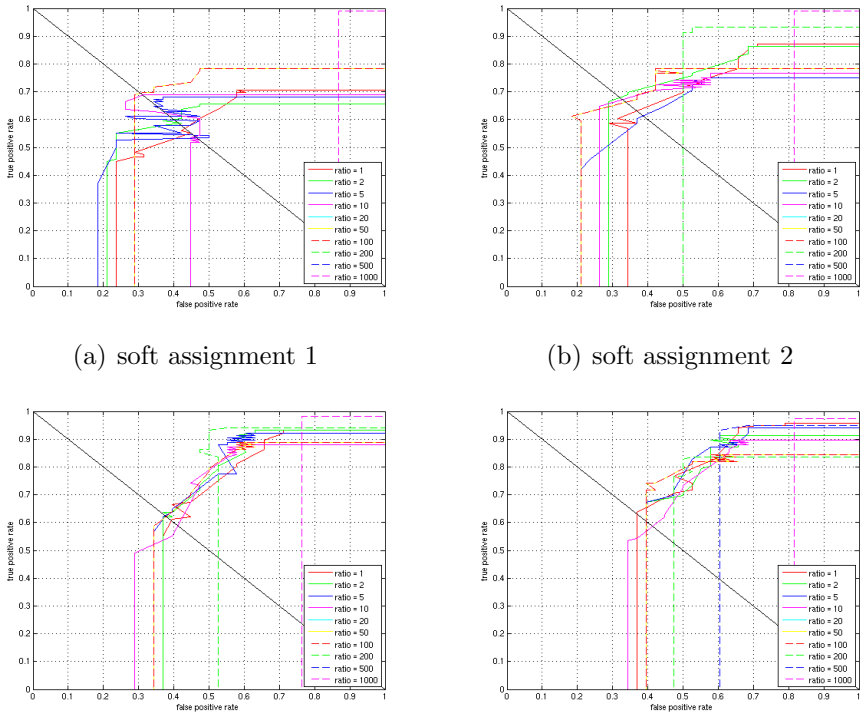
Table 3.8: *Information about the variation of the parameters. Here T means the number of AdaBoost iterations, support the different support values for the computation of the frequently occurring subgraphs, $soft\_assignment$ the soft assignments and ratio is the minimal value of $n_{pos} : n_{neg}$ for a subgraph to be used for the classification, n_{pos} and n_{neg} here are the number of subgraph embeddings in the positive training images and the negative training images, respectively.*

Figure 3.29: *Some examples of the activations of the cars-rear testset. The examples where taken for a support value of* 47.5%*, the edge-labeling method A and the soft assignment parameter* 1*. The color of the activated features gives the number of activations, where blue means that the feature is not often activated, and red means that the features is activated often.*

Computational performance

The bottleneck in the recognition pipeline based on graph mining turned out to be the subgraph-graph isomorphisms, which is an NP-complete problem. In contrast, for the itemset mining based method presented in the previous sections, matching of mined configurations could be done using sparse dot-products of vectors. Depending on the graph and the subgraph set the computation time for the subgraph-graph isomorphism varied between one second (for a support of 20% and a soft assignment of 1) and two hours (for a support of 5% and a soft assignment of 10). For the training of the classifiers the required time varies between some milliseconds and ten minutes, depending mostly on the number of AdaBoost iterations T and the number of frequently occurring subgraphs.

Overall, using graphs to express configurations of features is a compelling idea, and subgraph mining turned out to be a suitable method to detect some of the configurations common between instances of an object class. However, in the end the results were less interesting than those obtained with the simpler grid-based configu-



(a) soft assignment 1

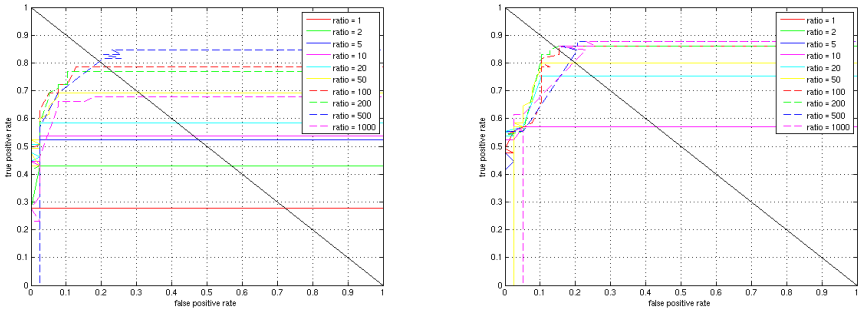
(b) soft assignment 2

(c) soft assignment 5

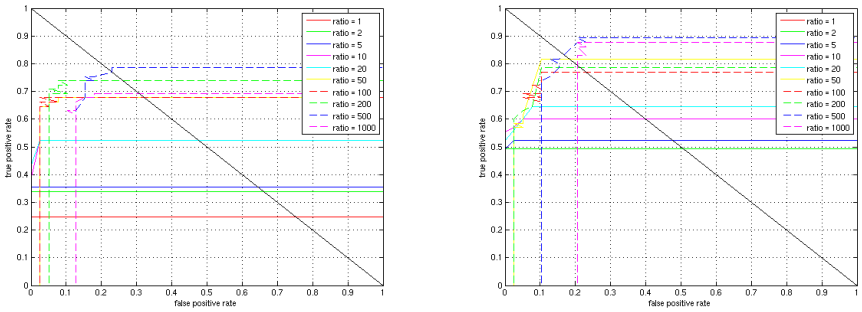
(d) soft assignment 10

Figure 3.30: *ROC curves of the motorbikes-side class. The curves were obtained by varying the parameter T from 1 to 50, setting the support value to 11% and letting the soft assignment parameter and the ratio constant. The curves for the same soft assignment parameter and different ratios are showed in the same plot.*

rations. Furthermore, extensive experimentation on additional datasets and deeper investigation of the effects of the parameters were inhibited by the computational demands of the graph-based methods.



(a) support 37%, soft assignment 1 (b) support 37%, soft assignment 2



(c) support 36%, soft assignment 1 (d) support 36%, soft assignment 2

Figure 3.31: *ROC curves of the cars-rear class. The curves were obtained by varying the parameter T from 1 to 50, setting the support value to 36% and 37%, setting the soft assignment parameter to 1 and 2 respectively and letting the ratio constant. The curves for the same support value, the same soft assignment parameter and different ratios are showed in the same plot.*

3.6 Related work

Our work relates to two strands of research: object recognition in computer vision, and data mining. From the data mining perspective, a few earlier works have tried to apply frequent itemset mining to visual data.

In [Tesic *et al.*, 2003] an extended association rule mining algorithm was used to mine spatial associations between five classes of texture-tiles in aerial images (forest, urban, pasture etc.). For this purpose the authors propose a modified APriori algorithm, which mines so called perceptual-associations, *i.e.* which types of tiles appear jointly in the data. This allows for analysis of the aerial image dataset characteristics, but not for any kind of object (-class) recognition. It is interesting to note, that the authors cluster the texture descriptors of the aerial tiles into what they call a "visual thesaurus", conceptually quite similar to the visual words that are so popular these days.

In [Ordonez and Omiecinski, 1999] a straight-forward application of association rule mining is used to identify jointly occurring geometric primitives as a means for object detection. However, the approach is only evaluated on quite simple artificial data and thus cannot be compared with the state of the art in object recognition.

In [Antonie *et al.*, 2003] association rules were used to create a classifier for breast cancer detection from mammogram-images. Each mammogram was first cropped to contain the same fraction of the breast, and then described by photometric moments. Compared to our method, both works were only applied to static image data containing rather small variations.

[Zaiane *et al.*, 1998] mines databases of annotated images using a diverse set of features such as keywords, file type, and global color and texture features. The focus is on finding hidden correlations between the different modalities of the data, rather than on the visual data itself.

From the object recognition perspective our work relates to several subfields. For our first contribution, mining specific objects from video, a large body of work obviously exists that deals with mining some kind of information from videos. However, few works have dealt with the problem of mining objects composed of local features from video data. In this respect, the closest work to ours is by Sivic and Zisserman [Sivic and Zisserman, 2004]. However, there are considerable differences. [Sivic and Zisserman, 2004] starts by selecting subsets of quantized features. The neighborhoods for mining are always of fixed size (e.g. the 20 nearestneighbors). Each such neighborhood is expressed as a simple, orderless bag-of-words, represented as a sparse binary indicator vector. The actual mining proceeds by computing the dot-product between all pairs of neighborhoods and setting a threshold on the resulting number of codebook terms they have in common.

While this definition of a neighborhood is similar in spirit to our transactions, we also include information about the localization of the feature within its neighborhood. Furthermore, the neighborhood itself is not of fixed size. For scenes containing significant motion, we can exploit our fast motion segmentation to restrict the neighborhood to features with similar motions, and hence more likely to belong to a single object. As another important difference, unlike [Sivic and Zisserman, 2004] our approach does not require pairwise matching of bag-of-words indicator vectors, but it relies instead on a frequent itemset mining algorithm, which is a well studied technique in data mining. This brings the additional benefit of knowing *which* regions are common between neighborhoods, versus the dot-product technique only reporting *how many* they are. It also opens the doors to a large body of research on the efficient detection of frequent itemset and many deduced mining methods.

Our extended method for mining frequent feature configurations for object-class type of data, has to be seen in a wider context of using spatial arrangements in object class recognition. The idea of using spatial configurations of local features is widely used in object class recognition. For instance, the *constellation model* [Fergus *et al.*, 2003] models the spatial arrangement of local features as a joint probability distribution. Inference in this fully connected model has high computational complexity and thus supports only a few features in practice. Fergus *et al.* thus suggest a simplified and more efficient star topology in [Fergus *et al.*, 2005].

Closer to our approach is the work of Lazebnik *et al.*, who propose semi-local arrangements of affine features for object detection [Lazebnik *et al.*, 2004]. Their method builds directly on features, without vector quantization, and starts by detecting geometrically stable triples of regions in pairs of images. The candidate pairs are summarized by a description which averages over their geometric arrangement. This description is validated on other examples and, if found repeatedly, used for recognition. Our approach instead, builds on vector-quantized features, defines a scale invariant tiled neighborhood, and employs established data mining techniques to find recurring neighborhoods. In addition to being computationally much more efficient, this allows for more variability in the feature appearances. We avoid searching over pairs of images, and mine the whole, large dataset globally *at once*.

Expressing configurations of features as graphs is a quite natural idea, and many works have built on it, *e.g.* the just mentioned [Fergus *et al.*, 2003]. However, graphs are a complex data structure, and thus, unlike [Fergus *et al.*, 2003], few works have proposed scalable algorithms. This was our motivation to look into graph mining algorithms for that purpose, as discussed in Section 3.5 of this chapter. Here, little work had been done, and only recently, in parallel to our work, [Nowozin *et al.*, 2007] have proposed some related methods. They propose learning a classifier based on boosting weighted substructures. The substructures are selected from the powerset of visual words in an image. In each iteration of the learning process, the most

relevant substructures are found using a graph mining or itemset mining method. In contrast to our mining methods, the heart of their system is the classifier, which uses itemset or graph mining at every learning stage as an optimization. Furthermore, unlike our method for mining frequent feature configurations, their itemset mining method does not encode any kind of spatial configurations of features.

3.7 Discussion and Conclusions

In the preceding sections we have applied itemset mining methods on a variety of object recognition tasks. This was motivated by the recently popular visual words, which allow us to encode an image as a set of items.

Based on this basic notion, we derived methods for video mining, for mining of feature configurations for object class recognition, and also evaluated alternative graph mining methods. In all of these areas we proposed methods, which consider local features not as an orderles set, but add an extra processing layer which encodes their spatial relationships prior to mining. This resulted in datasets with up to hundreds of thousands of transactions and up to millions of items.

We could show that itemset mining offers a suitable method to handle these large amounts of data and to find frequent patterns efficiently. Experiments for a video mining task showed, that our method is able to mine the most frequently occurring objects in music video clips. Our extended method for mining frequent feature configurations was evaluated in the context of object class recognition. We could show that the mined configurations of features have far higher discriminative power than individual features (Section 3.3.3). Moreover, the mined itemsets and rules exhibit the same pleasant properties as their counterparts in other fields: they can be analyzed and are easily interpretable by humans.

Motivated by these positive results, we combined our method with a very successful object-class recognition and localization method (the ISM framework of [Leibe *et al.*, 2008]), hoping, that the use of configurations over individual features would improve recognition performance. While only a simpler, un-optimised version of the ISM pipeline was used, the object localization results of the combined system were not better than the method using only single features as input. Qualitative observations showed, that the maxima in the voting space were less discriminative with configurations than with single features. This would lead to the explanation, that the ISM relies on an agglomeration of (possibly very weak) evidence from many sources, *i.e.* every single contribution to collect the maxima in the space counts. In the contrast, our method mines the bare essence of important feature configurations. Having less evidence makes the voting procedure more vulnerable to outliers.

A side path, which investigated graph mining lead to some interesting initial results, as the mined patterns showed the same semantic interpretability as the frequent configurations of visual words obtained using itemset mining. However, it turned out that the graph mining algorithms are by far not as scalable as itemset mining, especially for dense graphs, as they are very typical for a fully connected graph of all local features in an image. While we proposed some initial solutions (semi-local graphs based on k-NN or spatial neighborhood), further work towards optimizing the encoding and mining process would be required.

The probably largest disadvantage of the itemset and graph mining based approaches is that the outcome of the mining is sometimes rather uncontrollable and does not always behave "linearly", in the sense that small changes in parameters can lead to unexpected fluctuations of the results. It could be, that these flucations are exactly due to the limited amount of data usually available in the common benchmark datasets for object class recognition.

Several elements of our method could be further optimized. One possible extension would be to make matching of already mined configurations in novel images even more efficient using approaches inspired by FP-trees used in itemset mining algorithms such as FP-Growth. An interesting combination with another work would be to try to express configurations of local features as spatial pyramids. Similar to the spatial pyramid match kernels used in [Lazebnik *et al.*, 2006], but on a local level instead of a global level. The resulting "hierarchical configurations" could then again be treated with itemset mining methods to find interesting spatial patterns of local features.

Overall, while only the first few steps with itemset mining in databases of visual words have been presented here, we believe that the approach has further potential. This is for two reasons: philosophically, mining fits quite well between the two extremes of learning a model, or using a simple exemplar based approach. While popular methods methods such as pLSA are able to learn the hidden concepts [Sivic *et al.*, 2005], that make up an object(-class) they suffer from scalability. On the other extreme of the spectrum, exemplar-based approaches have been shown [Chum and Zisserman, 2007] to be a very straightforward and powerful approach for object class recognition, too. (Itemset) mining methods fit just between those: No hidden concepts can be learned as in pLSA, but the data is efficiently analyzed for the most essential patterns, neglecting irrelevant information. The latter is directly leads to the second reason: the availability of huge datasets. Enormous amounts of data (*e.g.* on the Internet) might lead to approaches which do not require any models any more, but efficient analysis of the data. Furthermore, thinking towards on-line learning, unsupervised learning, or on-line relevance feedback, methods are required, that answer every potential user query or intent sufficiently well. Methods, which require extensive training for each individual object (-class) to be recognized are not of much use in such a scenario.

4

Mining Objects and Events in large, multimodal Datasets

4.1 Introduction

In this chapter we take data mining in visual data to a higher level and to larger amounts of data. Instead of mining basic visual entities such as frequent feature configurations, we deal with the task of automatically detecting objects (such as landmark buildings) from large amounts of visual data on the Internet.

This task has to be seen in the context of the astonishing growth of the Internet in the last 10 years, both in terms of users and technical capabilities. Combined with the widespread use of digital cameras, this growth has led to the creation of large online databases of visual data, most notably community photo collections such as Flickr (`http://www.flickr.com`). These collections contain vast amounts of images, which pose both great challenges and opportunities to the computer vision researcher. While the large number of data items demands extremely scalable algorithms and systems, the collections also contain a great deal of multi-modal information with redundand descriptions across modalities, which is the key for unsupervised mining from the Internet.

Against the backdrop of the state-of-the-art object recognition, these developments allow us to deal with a crucial but often neglected building block towards Internet-scale image retrieval: the automated collection of a high quality image database with complete annotations. More precisely, from the large amount of sparsely labeled content in community photo collections, the task is to mine clusters of images containing objects in a fully unsupervised manner. For each mined item, we automatically derive a textual description and links to related content on the Internet. The resulting "cleaned" image database for the mined objects and events is of far higher quality than the original data and facilitates a variety of applications. For example, the mined entities can be used for automated annotation of photos uploaded to community collections, for retrieval and browsing of landmark buildings [Philbin *et al.*,

2007], automatic 3D reconstruction of landmarks [Vergauwen and Van Gool, 2006; Goesele *et al.*, 2007], or for tourist guide applications on mobile devices [Paletta *et al.*, 2006; Quack *et al.*, 2008; Takacs *et al.*, 2008] (where users can point the integrated camera of their device to a sight and retrieve information about it).

4.1.1 Outline of the chapter

In this chapter we demonstrate fully automatic, world-scale image mining from community photo collections.

We first introduce the most relevant sources for photos on the Internet (Section 4.2). We collect data from some of these sources and cluster the retrieved photos according to several different modalities (including visual content and text labels) and clustering strategies (Section 4.3.2).

For each cluster, we additionally calculate a set of cues, such as the number of different days the photos in the cluster were taken on, the number of users who took the photos, *etc.* We show how these additional features can be used to train a subsequent classifier, which determines if an image cluster represents an object or an event (Section 4.4).

We then apply Frequent Itemset Mining on the text associated with each cluster in order to assign cluster labels. We propose an algorithm that employs the resulting frequent itemset labels to link clusters to Wikipedia pages providing additional information about the cluster content, and that then in turn takes the Wikipedia entries to verify clusters and filter out false assignments (Section 4.4.2).

Closing the loop, we finally demonstrate how the verified clusters can be used to automatically label and geo-locate additional photos, for which no geotags were available in the first place (Section 4.5).

Results for all steps of the processing pipeline are then shown in Section 4.6.

4.2 Community Photo Collections on the Internet

Sharing information is one of the main purposes of the Internet. While in early years most of the published content consisted of text documents, technical advances led to the ability to share multimedia data such as photos and videos. The sharing aspect gained increasing attention and led to the formation of specific destinations on the Internet focussing on this key ability. Being able to share photos with friends and family is probably one of the most popular activities in this area. The sites

Type	Example	Count
Location/Travel	nyc, italy, trip	54
People	girl, baby,	7
Activity/Event	wedding, party, concert	14
Other/Abstract	animal, sky, red	67

***Table 4.1**: Tag statistics*

which offer these desired functionalities to the end-user are becoming a great pool of imagery for computer vision research. One of the most popular photo-sharing sites, Flickr, shall serve as an example to explain the available features and their use for our purposes.

Flickr (`http://www.flickr.com`) was founded in early 2004, just a few months before the research for this thesis started. As of end of 2007 it hosted more than 2 billion images. Some of the features relevant for our purpose include:

Tagging / Folksonomy: The term folksonomy [Mathes, ; Smith, 2004; Wal, 2005] stands for the concept of collaborative annotation of documents by the means of so-called tags. A tag is a keyword or term assigned to a piece of information, *e.g.* a text document or an image. In contrast to classical annotation, tagging does rely neither on a controlled vocabulary nor on specially trained editors. In other words, anyone can assign any word to a given piece of data. Figure 4.1 shows the most popular tags on flickr. This approach simplifies annotation for the annotating person drastically. On the other hand, classic problems such as synonymy, polysemy or imprecise descriptions due to the typically very short single-word tags are not dealt with. While the resulting annotation of data is of far lower quality than its counterparts created by trained professionals for traditional archives, they are much more precise than for instance the text of a web-page an image is embedded in. In fact, the large user base (up to millions of users) contributing to the tagging efforts leads to an increased probability that a data item referring to the same content is indeed frequently labeled with the same tag. For the vision research community photo collections with tagging capabilities thus provide access to a large database of images with weak annotations. Note that the most popular tags in Figure 4.1 also give an insight on the type of pictures uploaded to Flickr. Table 4.1 shows statistics for the most popular tags (as of September 2008). As can be seen, a large fraction of photos is related to some location or travel. Many of these photos will contain pictures of touristic sights such as landmark buildings, and are thus optimal for our endeavor.

Geotagging: Geotagging is a special form of tagging, where a piece of data is labeled with a geographic location it is related to. For image databases this typically is

***Figure 4.1**: Most popular tags on Flickr. The size of the text is proportional to the tag's popularity.*

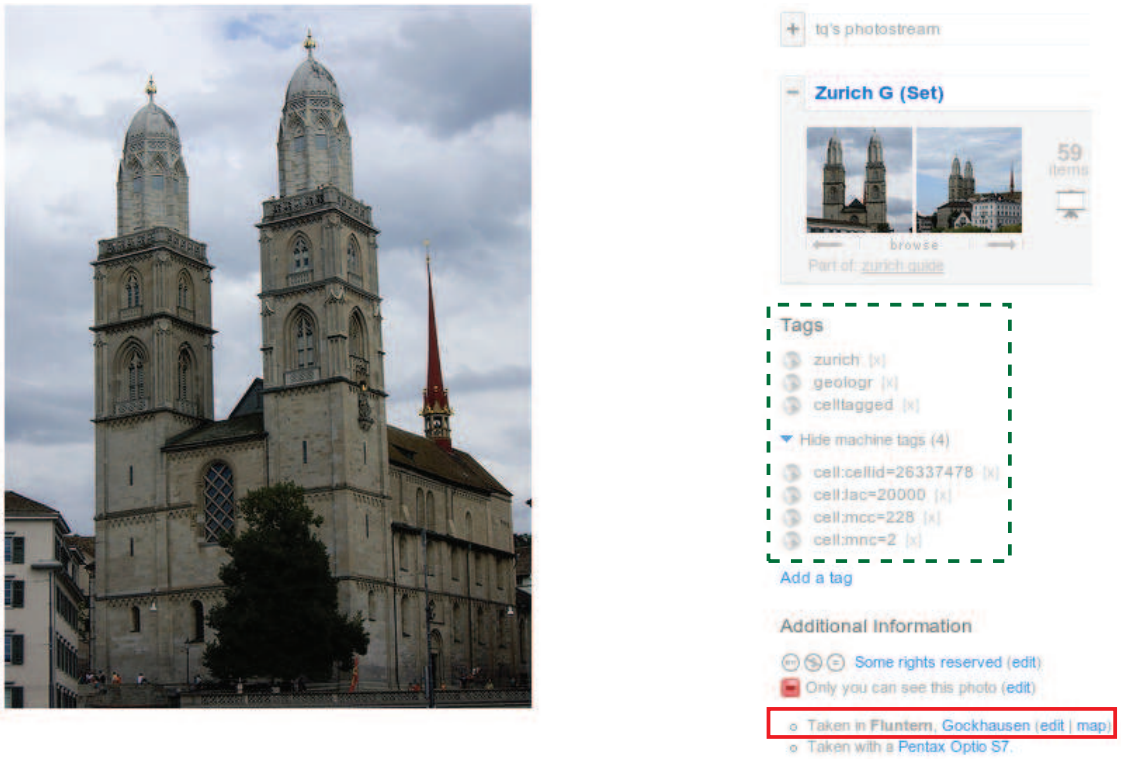


***Figure 4.2**: Tags and geotags on Flickr. Dashed Box: Textual tags, and machine tags (describing the mobile network location in form of a Cell Global Identifier). Solid Box: Location name based on GPS coordinates.*

the location a photo was taken at. Geotagging can be provided in several forms, the most precise being longitude and latitude values obtained from a Global Positioning device (GPS). Further possibilities are annotation with a mobile phone cell tower identifier (CGI, Cell Global Identity, see Chapter 5.2 for a detailed description), manual assignment to a postal address, or placing the item manually onto a digital map. Flickr introduced geotagging officially in August 2006. Users can provide GPS locations with their photos or drag them to a map manually. Furthermore, so called "machine tags" allow users to provide information about mobile network cell-tower ids in a special tagging format, Figure 4.2 shows an example. Flickr reports that over 2 million such geotagged photos are currently uploaded each month. As we will discuss below, we make extensive use of this geotagging information.

API: In context of the "Web 2.0" movement, many on-line platforms offer access to their data and services by means of an Application Programming Interface (API) usually implemented as a web-service. For instance, Flickr allows querying their database of photos using several criteria, e.g. by tag, by time, by user or by geographic location. The list of returned photos for an API call allows to download the image itself and related meta-data such as tags, descriptions, user comments, or geo-information. Having this kind of access allows us to integrate databases such as Flickr in our software easily.

Besides Flickr, the following photo-sharing sites may also be of interest to the vision researcher:

Panoramio.com: Panoramio focusses on geotagging. All photos on the platform are geotagged, however, the only available textual description is a title, *i.e.* no tagging functionality is offered to end-users. Like Flickr, Panoramio also offers an API. Most photos on Panoramio are travel related photos.

Facebook.com: Facebook is a digital community with a focus on social life and student life, rather than photo sharing. However, the integrated ability to upload fotos quickly made it the largest photo-sharing web-site in the world. The stumbling number of 24 million photos are uploaded to Facebook – daily. The photos on facebook cover a wide range of topics. The largest fraction seems to cover events, people *etc.* due to the platform's focus on social networking rather than photography.

Picasa Web Albums: Picasa is a desktop photo management tool provided by Google. Users can export their albums to their Google account to share the albums. While several features including tagging and geotagging are available to users, at the time of writing the platform seems to be a conglomerate of individual web-albums

rather than a large pool of photos with global search and sharing abilities. An API to the Picasa web-albums is available.

In this work we focus on data from Flickr. While it is not the largest pool of photos on the Internet, it is the one with the best quality of data and content.

4.3 Mining Clusters

Our approach is based on photographs which have been tagged with their geographic location. This allows us to mine the world in a scalable manner without any prior knowledge on landmarks and their locations. To that end, we partition the world into a grid of square tiles and retrieve for each tile all the corresponding geotagged photos from Flickr. The geographic tiling allows us to handle the size of this vast problem and to parallelize computations.

4.3.1 Gathering the data

To gather the raw data, we query community photo collections such as Flickr. First, we divide the earth's surface into square tiles T_k of about 200m side length. A tile center is set every 100m (in longitude and latitude direction), such that the tiles have a high overlap. For each tile, we query the Flickr API with the tile's center coordinates and bounding box to obtain all geotagged photos for that area. Figure 4.3 shows a section of a map with the tiles used for querying overlaid. In total, we processed about $70'000$ tiles for this work, covering several European urban centers, namely Paris, Rome, Venice, Oxford, Zurich, Pisa, Munich, Tallinn, Prague, and St. Petersburg. Table 4.2 lists the urban areas we covered and the number of tiles and photos retrieved for each area. In total, we covered an area of about 700 square kilometers. The majority of tiles (about $52'000$) were empty. The remaining tiles contained on average 10 and a maximum of 3750 photos. For each photo we downloaded, we also obtained the associated metadata, namely the textual descriptions (tags, title, description), user-id, and timestamps.

4.3.2 Photo Clustering

Once the photos for each tile have been downloaded, we process each cell to find clusters of photos with similar content as object candidates. We first create dissimilarity matrices for several modalities (visual and text) by calculating the pairwise distances between photos for each modality. A hierarchical clustering step on the dissimilarity matrices then creates clusters of photos for the same object or event. In the following we discuss the features and distances used for each modality.

***Figure 4.3**: Tiles over Paris. The size of a tile is marked in red. Note the overlap of 50% (100m horizontally and vertically).*

Visual Features and Similarity

To identify pairs of photos which contain the same object, we employ matching based on local, scale invariant features and projective geometry. We first extract visual features from each photo. For this, we employ again SURF [Bay *et al.*, 2006b] features due to their fast extraction times and compact description shown in earlier works. Each image is thus represented as a bag of 64-dimensional SURF feature vectors. For each pair of images in a tile T_k, we find matching features by calculating the nearest neighbor (NN) in Euclidean distance between all feature pairs, followed by a verification with the $2nd$ nearest neighbor criterion from [Lowe, 2004]. Note that this linear matching procedure is fast enough, since the problem is separated into the geographic tiles. Using scaleable indexing methods such as the ones discussed in Chapter 6 could lower the processing times of the system even further, while slightly compromising matching precision.

To find object candidates from the matching features we next calculate homography mappings for each matched image pair $\{i, j\}$ [Hartley and Zisserman, 2004]

$$H\mathbf{x}_n^i = \mathbf{x}_n^j \ , \quad n \in 1 \ldots 4 \ , \tag{4.1}$$

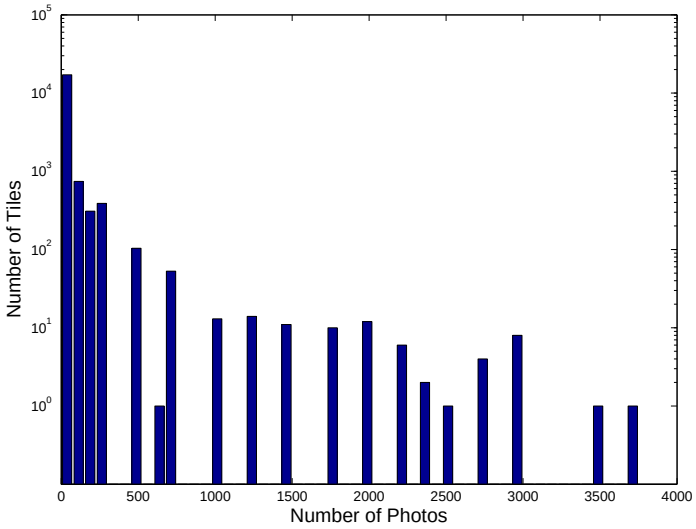


***Figure 4.4**: Number of photos per tile (log scale).*

Name	# tiles	#photos	area (km^2)
Munich	18'228	24'069	184.99
Oxford	2'112	7'431	22.05
Paris	12'532	87'452	127.57
Pisa	723	1'950	7.78
Prague	11'110	28'872	113.22
Rome	14'397	48'750	146.38
St. Petersburg	3'400	2'573	35.18
Tallinn	890	1'350	9.51
Venice	449	7'708	4.92
Zurich	5'663	12'602	58.15
Total	**69'504**	**222'757**	**709.74**

***Table 4.2**: Urban areas processed in this work and the number of tiles and photos per area.*

where H is the 3×3 homography whose 8 degrees of freedom can be solved with four point correspondences $n \in 1 \ldots 4$. To be robust against the aforementioned outliers, we estimate H using RANSAC [Fischler and Bolles., 1981]. The quality of several estimated models is measured by the number of inliers, where an inlier I is defined by a threshold on the residual error. The residual error for the model is determined by the distance of the true points from the points generated by the estimated H. We accept hypotheses with at least 10 inliers I as a match.

Using this kind of homography mapping works well in our case, since we have many photos taken from similar viewpoints. A fundamental matrix could handle larger viewpoint changes, but it is also more costly to compute, since it requires more inliers to find the correct model. Furthermore, mapping planar elements (such as building facades) works very well with homographies. An example is shown in Figure 4.5. In spite of the strong viewpoint change, a reasonable homography mapping could be found in this example, while most of the true outliers are removed. A similar approach (using affine transformations estimate from single affine covariant features) has also been successfully applied in [Philbin *et al.*, 2007] for a retrieval engine on a database of landmarks from Oxford handling astonishing viewpoint and scale changes. As mentioned above, the accuracy achieved with these kinds of visual features is far better than with any kind of global features, which are still often used for mining and retrieval in visual databases.

The distance matrix is built from the number of inlying feature matches I_{ij} for each image pair, normalized by the maximum number of inliers found in the whole dataset.

$$d_{ij} = \begin{cases} 1 - \frac{I_{ij}}{I_{max}} & \text{if } I_{ij} \geq 10 \\ \infty & \text{if } I_{ij} < 10 \end{cases} \tag{4.2}$$

In our implementation we set $I_{max} = 1000$, since we extract at most 1000 SURF features per image (sorted by their discriminance), *i.e.* the distance d_{ij} ranges in $[0 \ldots 0.99]$. Figure 4.6 shows the distribution of distances for all pairs with distance $d_{ij} <$ inf in the dataset. It can be observed, that the majority of pairs have a rather large distance, corresponding to 10-20 inlying feature matches.

Text Features and Similarity

Three sources for text meta-data were considered for each photo downloaded from Flickr: tags, title, and description. We combine these three text fields into a single text per photo for further processing stages. The first stage consists of a stoplist. In addition to the common stopwords, this list also contains collection-specific stopwords such as years, months, and terms such as "geotagged", "trip", "vacation", "honeymoon", *etc.* Furthermore, from each photo's geotag we know its location and (through reverse geocoding [Lewis *et al.*, 2007]) the corresponding place name,

***Figure 4.5**: Feature matching with Homography. SURF feature matches are shown in red, inlying matches for the estimated homography in green.*

for instance "Rome, Italy". These location-specific place names were added to the stoplist for each photo depending on its geotag. Filtering terms with these custom stop-lists turned out to be crucial to obtain good cluster labels in later processing stages.

As with the visual features, we proceed by calculating the pairwise text similarities between the documents (photos). A vector space model with term weighting of the following form is applied:

$$w_{i,j} = L_{i,j} * G_i * N_j$$

Note that in the standard $tf * idf$ ranking [Salton and McGill, 1986] $L_{i,j} = tf_{i,j}$, $G_i = \log \frac{D}{d_i}$ and $N_j = 1$, where $tf_{i,j}$ is the frequency of term i in document j, d_i is the

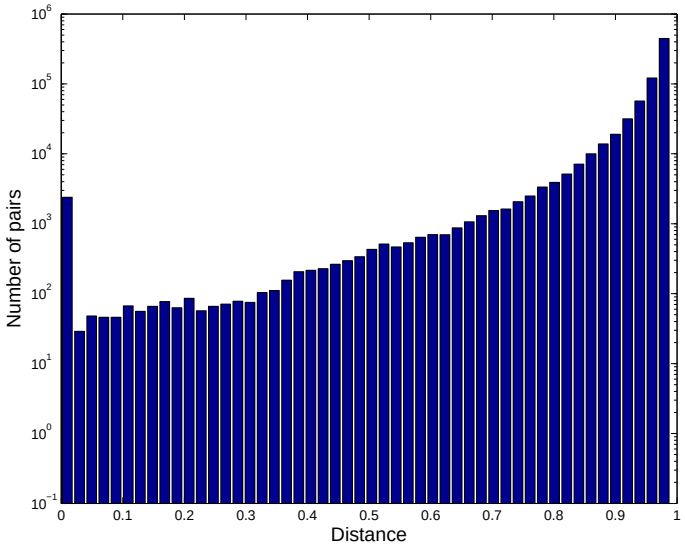


***Figure 4.6**: Histogram of visual distance values (log scale).*

number of documents containing term i, and D is the total number of documents. In our system, the weighting elements are as follows

$$L_{i,j} = \frac{\log(tf_{i,j}) + 1}{\sum_j \left(\log(tf_{i,j}) + 1\right)} \tag{4.3}$$

$$G_i = \log\left(\frac{D - d_i}{d_i}\right) \tag{4.4}$$

$$N_j = \frac{U_j}{1 + 0.0115 * U_j}$$

where U_j is the number of unique terms in document j. The rationale behind the modifications of the weighting terms over the standard $tf * idf$ are as follows. The logarithm in $L_{i,j}$ adjusts/dampens weights of multiple occurring words per document. G_i is a probabilistic inverse document frequency as proposed in [Croft and Harper, 1997], which, unlike idf, assigns negative weights to terms that appear in more than half the documents. Finally, the additional term N_j is a pivoted unique normalization which is used to correct for discrepancies in document lengths [Singhal *et al.*, 1996]. We use the mySQL (`www.mysql.com`) full-text search, which can be configured to use the modified $tf * idf$ ranking, to compute the text distance matrix for the photos belonging to each geographic grid tile.

Additional Features

Besides the visual and text similarities between photos, we also considered several additional cues. We store the timestamps and the user data (*i.e.* the Flickr user,

	Visual	Text
Single-link	0.985	0.989
Complete-link	0.99	0.99
Average-link	0.99	0.99

***Table 4.3**: Cut-off distances for clustering*

who took or uploaded a photo). As we will show below, these cues allow us to classify each cluster candidate into event or object types.

Clustering

For each tile T_k, we apply hierarchical agglomerative clustering [Webb, 2002; Jain and Dubes, 1988] to the distance matrix of each modality. This clustering approach was chosen, since it builds on a dissimilarity matrix and is not restricted to metric spaces. It is also rather flexible and very fast, once the full distance matrix is available. Using different linking criteria for cluster merging allows us to create different kinds of clusters. We employed the linkage methods described in Chapter 2.2, namely single-link, complete-link, and average-link.

The motivation behind these measures is to capture different kinds of visual properties that allow us to associate a semantic interpretation with the resulting clusters. Single-link clustering adds images to a cluster as long as they yield a good match to at least one cluster member. This results in elongated clusters that tend to span a certain area. As a result, if visual features are the basis for clustering, this procedure can group panoramas of images that have been taken from the same viewpoint, or series of images around an object. In contrast, complete-link clustering enforces that a new image matches to all cluster members. This strategy will therefore result in very tight clusters that contain similar views of the same object or building. Average-link clustering, finally, takes a compromise between those two extremes and provides clusters that still prefer views of the same object, while allowing more flexibility in viewpoint shifts. In our approach we do not want to restrict ourselves to any single of those alternatives; instead, we pursue them in parallel. Such an approach makes it possible to derive additional information from a comparison of cluster outcomes. For example, we may first identify distinct objects or landmark buildings through complete- or average-link clusters and later find out which of them are located close to each other by their membership in the same single-link cluster. Table 4.3 summarizes the linkages and cutoff-distances used for each modality.

***Figure 4.7**: Class examples: object, event, none.*

4.4 Labeling Clusters

In the preceding sections, images with similar content or annotations were grouped into clusters, which ideally should depict a single entity. In this section, the goal is to look into the contents of the clusters in more detail. First, we classify the clusters into objects and events. In a next step, we derive textual labels for the clusters from the associated metadata. Furthermore, we introduce an approach to formulate text queries from the labels, which are submitted to Wikipedia to assign articles to the clusters. A final verification step uses the images found in the Wikipedia articles to verify this assignment.

4.4.1 Classification into Objects and Events

To discriminate between objects and events, we rely on the collected metadata for the photos in each cluster. An "object" is here defined as any rigid physical item with a fixed position, such as landmark buildings, statues, *etc.* As "events", we consider occasions that took place at a specific time and location, for instance concerts, parties, *etc.* Thus, we include as features f_1, f_2 the number of unique days the photos in a cluster were taken at (obtained from the photos' timestamps) and the number of different users who "contributed" photos to the cluster divided by the cluster size.

$$f_1 = |D| \tag{4.5}$$

$$f_2 = \frac{|U|}{|N|} \tag{4.6}$$

where $|D|$ is the number of days, $|U|$ the number of users, and $|N|$ the number of photos in the cluster. Typically, objects such as landmarks are photographed by many people throughout the year; an event on the other hand usually takes place only at one or two days and is covered by fewer users. Note that we only consider clusters with $N > 4$ here. We manually labeled a ground truth of about 700 clusters

with the class labels "object", "event", and "none". See Figure 4.7 for an example of each class. We then trained an individual ID3 decision tree [Quinlan, 1986] for the classes "object" and "event" on half of the labeled data and used the other half for validation. The task in training and testing was to discriminate the target class ("object" or "event") against all other classes. Cross-validated over 10 random data partitions, this simple classifier was able to achieve 88% precision for objects and 94% for events with a standard deviation of 0.07% and 0.04%, respectively. (In fact, due to the only two features considered we deal here rather with a decision stump than with a decision tree.)

4.4.2 Linking to Wikipedia

Having the clusters classified into objects and events, the next processing layer intends to add more descriptive labels. The goal is to not only label the clusters with the most dominant words, but also to automatically link them to content on the Internet, such as corresponding Wikipedia articles. Such a solution allows auto-annotation of unlabeled images, even down to outlining object-parts using the information from other pictures of the same entity. A recognition service building upon our labeled database could then match the query to the corresponding database entry and return the assigned Wikipedia content to the user. Such systems have been proposed before (*e.g.* [Paletta *et al.*, 2006; Quack *et al.*, 2008]), but the automatic collection of the database from user-generated content has not been addressed yet.

The proposed approach first finds relevant word combinations from the text associated with each cluster using a frequent itemset mining algorithm. The resulting frequent combinations are then used to query Wikipedia in a second step. An image based matching step finally verifies that the links are indeed correct.

Frequent Labels

Flickr and similar community photo collections provide us with text associated to photos. However, the text is often noisy, and not all images are labeled. Furthermore, if we want to use the text to find out more about the object by querying Internet search engines, we need to create queries from the raw tags. Any combination of words from the text could be the "correct" query. However, finding and trying all possible combinations would mean considering 2^N combinations of words, where N can easily be in the hundreds. We therefore resort to frequent itemset mining (see Chapter 2.4) to find the most frequent combinations of words efficiently. Those can serve both as labels for the objects and as query input for the next stage.

In our setting, the text associated with each photo (tags, caption, titles, *etc.*) generates a transaction, and the database consists of the set of photos in a cluster.

We use an implementation [Borgelt, 2005] of the *FP-Growth* algorithm to mine the frequent itemsets for each cluster, using a minimal support threshold of 0.15 (*i.e.* 15%). In order to ensure scalability, only the top 15 itemsets per cluster are kept.

The advantage of using itemset mining over other probabilistic method is its speed and scalability. Tens of thousands of word combinations can be processed in fractions of seconds. Furthermore, mining variants such as maximal or closed frequent itemsets, as well as additional statistical tests on the sets, offer further opportunities for optimization (see Chapter 2.4). For instance, maximal frequent itemsets (itemsets with no frequent superset) are especially useful for human-readable labels on clusters, since their subsets are not listed as additional labels.

Querying Wikipedia and Link Verification

We use each frequent itemset mined in the previous section to submit a query to an Internet search engine. More specifically, we query Google (`www.google.com`), limiting the search to `wikipedia.org`. By doing so, the search covers Wikipedia in all available languages, so terms in different languages can be handled automatically. For each result list, the top 8 results are kept. Note that in the worst case, this generates $15 * 8 = 120$ possible URLs per cluster. We keep a score for each page, which counts how often the same page was retrieved using different queries. Next, we crawl each of the URLs and parse the corresponding Wikipedia page for images. The idea is now to use the Wikipedia content to verify the proposed linking between the cluster and the Wikipedia page. Chances are high, that our clusters contain some images taken from similar viewpoints as the ones used in Wikipedia. Thus, we extract features from the Wikipedia images and try to match them to all images in the cluster using the same method as described in Section 4.3.2. If we find a matching image, the proposed link is kept, otherwise it is rejected. Figure 4.8 visualizes the individual steps in linking clusters to Wikipedia content. The tags for the cluster (a) are mined to create frequent itemsets (b). Note how the proximity to the Louvre introduces noisy words such as “museum”, and how the expression “arc du triomphe” could refer also to the other, larger Arc Du Triomphe in Paris. The frequent itemsets (b) are fed as queries to Google, and the candidate URLs (c) are retrieved. The URLs are ranked according to how many queries had the corresponding URL in their result list. For each URL, the HTML of the corresponding Wikipedia page (d) is parsed to extract images. The images contained in the page are downloaded and matched back to the images in the cluster. Figure 4.8(e) shows the best match from the cluster with the image from the Wikipedia article (d). If such a match can be found, the corresponding Wikipedia URL is selected as verified annotation (f).

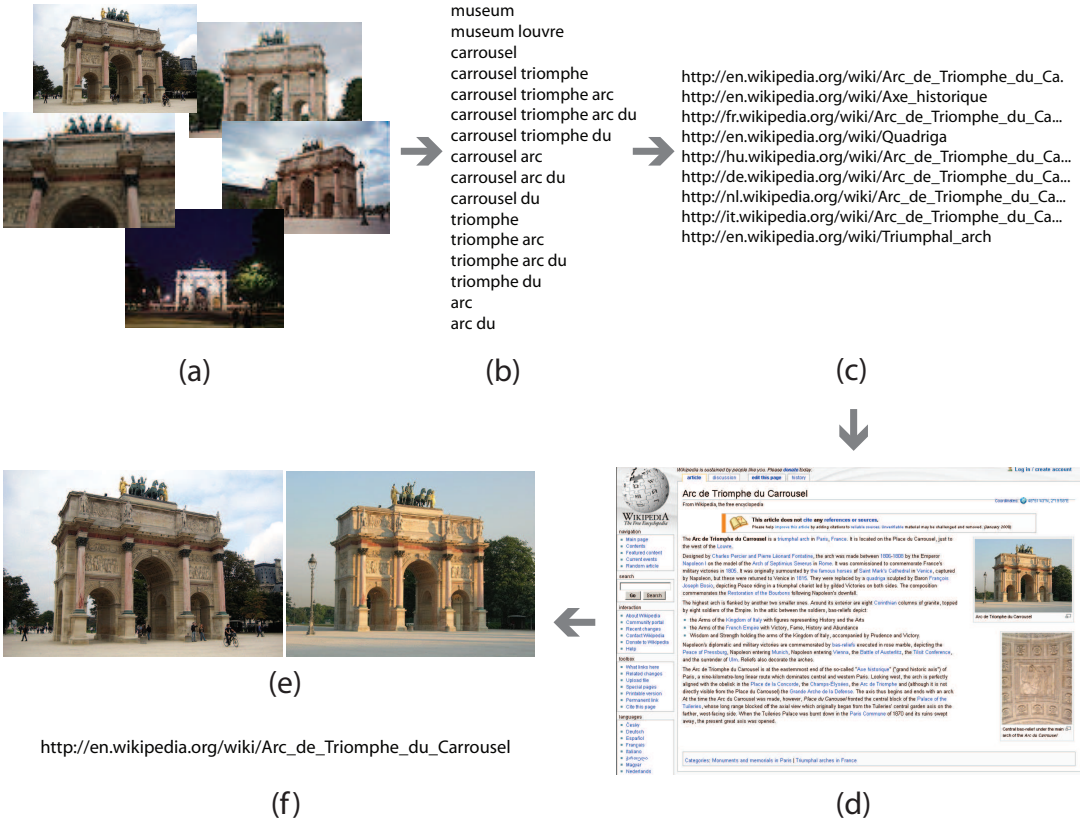


***Figure 4.8**: Matching clusters to Wikipedia articles. The text for the photos in a cluster (a) is mined for frequent word combinations (b), which are used to search Wikipedia for candidate URLs (c). Each image of an article (d) is in return matched to the images in the cluster. If a correct match (e) can be found, the candidate link is selected (f).*

4.5 Object-level Auto-Annotation

Using the data that was collected during the mining process, we can now annotate novel images and even refine the annotations for database images in several ways:

- Auto-tagging with most confident tags per cluster.
- Assigning related Wikipedia articles to images.
- Placing images (without geotags) on a map.
- Object-level annotation with bounding boxes around the objects.

In the following subsection we describe how we estimate bounding boxes for objects to achieve object-level annotations.

4.5.1 Estimating Bounding Boxes for Objects

Each of the mined object clusters was created by clustering images based on their pairwise distances as described in Section 4.3.2. The pairwise distances were calculated based on the number of (inlying) local feature matches for the image pairs. In other words, each image in a cluster matches to several other images showing the same object. We can now use these multiple cross-matches between images and derive an object-specific feature confidence value. In spirit this is similar to the feature confidence values calculated in Section 3.3, but for specific objects instead of object classes. A bounding box can then simply be estimated around the most confident features.

More specifically, the object-specific confidence value for feature f in image i is simply calculated as the number of inlying feature matches stemming from all other images:

$$c^o_{if} = \parallel \forall\ f\ \mid f\ \in I_{ij} \parallel\ \ j = 1 \ldots N^o \tag{4.7}$$

where f indexes the features in image i, and N^o is the number of images in the current object cluster o.

The estimation of the bounding box is based on a threshold on that confidence value, where the threshold t^o_i for object o in image i is defined as

$$t^o_i = min\left(t_{min}, \alpha * \frac{\sum_{f=1}^{M_i} c^o_{if}}{M_i}\right),\ \ M_i = \parallel f \mid c^o_{if} > 0 \parallel \tag{4.8}$$

where t_{min} and α are parameters with typical values $t_{min} = 2, \alpha = \frac{1}{3}$. The bounding box is drawn around all features with confidence higher than t^o_i, in other words around all features that have a confidence higher than a fraction α of the mean confidence value. Examples of the resulting confidence values and estimated bounding boxes are shown in Figures 4.9 and 4.10. The colors reflect the confidence value, the higher the value the brighter the color. In most cases, features which are selected as confident are very well located on the objects with few outliers. The occasional outliers typically receive a substantially lower vote. Thus, the simple bounding box estimation based on a threshold on the vote is sufficiently precise in most cases.

Note, that a more sophisticated localization of a bounding box by using a method such as Hough voting [Leibe *et al.*, 2008; Lowe, 1999] is not straightforward here: since we don't have training images of the objects, we don't know its extent and the location of features in relation to its center. Learning such a representation automatically from the mined data is left as future work.

These bounding boxes do not only allow object-level annotation, they can also improve indexing of features for the corresponding objects: only the features lying within the bounding boxes need to be considered for indexing. This will lower the signal-to-noise ratio in an index built on top of the mined data.

Figure 4.9: *Object-specific feature confidence values and bounding boxes. Part I: St. Peters Basilica, Rome.*

Figure 4.10: *Object-specific feature confidence values and bounding boxes. Part II: various examples of Paris sights).*

# Images	222'757
Size Metadata	1.1 GB
Size Features	111 GB
# Images assigned to clusters	73'236
# Verified Wikipedia Links	861
# Images in clusters linked to Wikipedia articles	15670
# Distances computed	217'330'144
# Distances $< \infty$	751'457

***Table 4.4**: Dataset statistics*

4.6 Experiments and Results

In the following, we present results on the whole dataset collected to this date, stemming from the 70′000 geographic tiles that were inspected by our algorithm. We first give an overview over the dataset, followed by subsections discussing the results of the individual processing layers. Table 4.4 summarizes the dataset statistics. In total over 220′000 images were downloaded from Flickr, their visual features amounting to 111 GB, and their metadata (tags, geotags, EXIF data *etc.*) to 1.1GB. Over 200 million pairwise distances had to be computed, less than 1 million was smaller than infinity. (Note that without the geographic tiling, we would have had to calculate over 20 billion pairwise similarities). In the end, a little over 73′000 photos could be assigned to a cluster.

4.6.1 Clusters

Here, we present results for different types of clustering. We start with a specific example to give an impression of the results we found. Figure 4.11 shows examples from the area around the Pantheon in Rome, and Table 4.5 summarizes numerical data for this example. The corresponding tile is among those with the largest number of elements, containing 2′250 images (several tiles overlap at this location; we report the numbers for the dominant one). It is well visible how the clustering splits the data into several semantically separate objects and contexts. For example, indoor (a) and frontal outdoor views (b) of the Pantheon are found as separate entities. Both contain a large number of photos: 546 and 481, respectively. Smaller clusters describe more specific elements, such as the view from the Pantheon onto the piazza (e), the obelisk situated behind the Pantheon (c), and even the tomb of Victor Emmanuel II (d) inside the Pantheon. Calculating the mean of the photo locations in each cluster allows us to place the cluster on a map. Clearly, the locations of the different clusters are estimated very close to the true positions of the corresponding entities. The clusters shown in this figure were obtained using single-link clustering.

Figure 4.11: *Clusters found around the Pantheon and the number of photos contained in each. Note the automatic separation into indoor (a), outdoor (b), and panorama views (e), and the discovery of separate objects (c,d). Mean locations of the photos are shown on the map. (e) is estimated at about the same position as (b) and is therefore not drawn on the map.*

Note how especially for clusters (a), (b), and (c), this allows us to merge a wide variety of views of the same object, since only the closest matching pair has to be connected by a distance smaller than the threshold. In total 27 clusters were found in this area, with a mean size of 75 photos. We evaluate clustering accuracy in terms of the cluster precision, *i.e.* the number of correct images divided by the total number of images in the cluster. As "correct", we count every image which contains the object the cluster refers to. If there are special contexts, such as an indoor view for an object, only those (*e.g.* indoor views) are counted as correct. Given that definition, the mean precision of the 10 largest clusters is over 98%. Note that since we deal with an unsupervised mining problem, we cannot give reliable results for recall. Qualitatively spoken, however, recall values will not be as impressive,

# Images in tile	2250
# Clusters	27
Cluster mean size	75
Cluster max size	546
Cluster min size	4
Mean precision 10 largest clusters	98%

***Table 4.5**: Summary of Pantheon Results*

since it may happen that photos of one object are spread over multiple clusters. Furthermore, it would be unrealistic to expect exhaustive recall from un-controlled databases such as Flickr.

For comparison, we also ran a clustering based purely on text, using all text similarities between the photos in this area. Depending on the parameters, we were only able to get 1-3 clusters with a precision of about 60%. Not only were we not able to discriminate between indoor and outdoor views based on text features, the clusters also contained many outliers which did not contain the relevant object at all. For instance, only 116 of the photos in the area carry tags such as "inside" or "interior", making a discrimination based on text very difficult. In contrast, cluster (a) in Figure 4.11 contains over 500 photos of the inside of the Pantheon. (The word "Pantheon" appears with 1′245 photos). Also in comparison to [Simon *et al.*, 2007], we are able to retrieve larger clusters while maintaining high precision.

To examine the results of the different types of visual clustering further, consider another example shown in Figure 4.12. It depicts the area around the Louvre in Paris. Figure 4.12(a) shows the estimated mean positions of single-link clusters. In total, the area is covered by 176 clusters; the largest cluster contains 418 elements, the mean size is 17 elements. One of the clusters (marked in yellow) is shown in Figure 4.12(b). Here, each pin represents the location of one photo. Note how strongly the positions vary. Some examples of the clusters' contents are shown in the column next to the map, again visualizing the mentioned variability in viewpoints. In contrast, Figure 4.12(c) shows the complete-link clusters for the same area. The more restrictive clustering criterion results in smaller and more compact clusters; the mean size is only 4 elements, and the maximum is 5. 207 complete-link clusters were found for this region; again one cluster is selected and its elements are shown in Figure 4.12(d). Their locations are more compact, and the contents of the cluster have less variability, as the examples next to the map demonstrate. Also note again the grid overlaid on the maps in (a) and (c), which shows the tiles we used to retrieve photos by their geotags (again, 4 cells make up a tile).

The results for average link clustering turned out to be quite similar to the ones obtained with complete-link clustering and are not shown here.

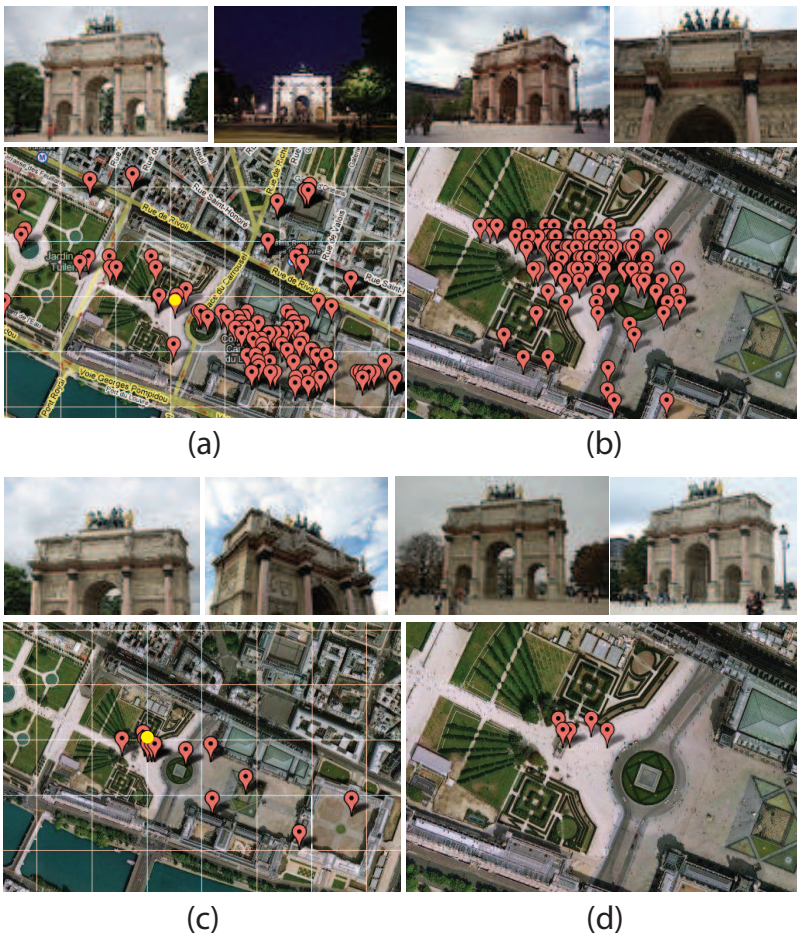


***Figure 4.12**: Clusters around the Louvre: (a) shows single-link clusters, the photos of the cluster marked in yellow are located as shown in (b). (c) shows complete-link clusters for the same area, again with the photos of the yellow cluster in (d). (Only clusters with at least 4 elements are shown).*

4.6.2 Objects and Events

The classifier described in Section 4.4.1 allows us not only to detect objects, but sometimes even events. Applying the ID3-tree to the entire dataset resulted in the following distribution of objects and events: of 6′511 clusters (single-link), 4′315 were classified as objects, 719 as events. Visual inspection on randomly picked clusters showed that the classification precision is very accurate, similar to the results obtained on the validation set in Section 4.4.1. Figure 4.13 shows some examples of event clusters. The first cluster contains images from 3 different events in a series taking place on different days ("Oxford Geek nights") and was recognized due to the same location it took place in. The second (a movie premiere in Italy) and third event (an exhibition in a gallery in Paris) were both covered by two different

Figure 4.13: *Typical events mined by our methods.*

photographers. The last line represents the majority of events: an event from a single day, covered by only one photographer.

An example where our simple classification into objects and events fails, is shown in Figure 4.14. The example shows a window located in the Vatican in Rome, from which the pope addresses the people. The clustering identified this particular window, and the labels refer to the pope. However, due to the distribution over several dates in several years, the cluster was classified as object and not as event. However, in this particular case the classification is hard to define: is it the place, the event, or the person that define the clusters' semantics?

Objects and event clusters can also be visualized on a map, as shown in Figure 4.15. Different classes of clusters are represented by different pin colors, the pin location is set at the mean position of the geo-tags of the images in the cluster. The example shows the area around the Sacre Coeur in Paris. Again, most of the clusters correspond to objects, only a few of them are events.

The smaller number of event clusters can be explained by two factors: relying mostly on visual cues, we can only detect events which take place in an environment where

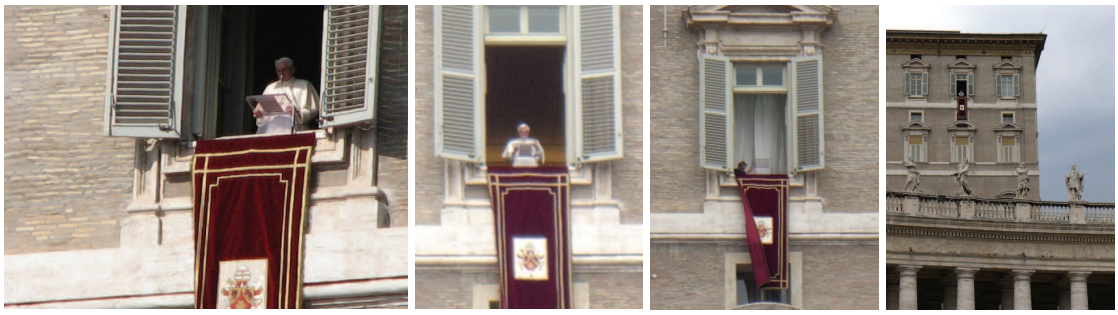


***Figure 4.14**: Misclassification (?) example. The Pope's window in Rome, labeled as object. The textual labels derived from the tags are* xvi, popebenedictxvi, benedict, stpeters, pope

the background matches between photos. Second, it seems that so far, in general fewer people geotag photos of events.

4.6.3 Multimodal Linking to Wikipedia

Figure 4.16 shows some typical results for the multimodal linking to Wikipedia. Each result is represented by a pair of images: the left image was extracted from Wikipedia, the one on the right is its closest match in the cluster (there are typically many more matching images in each cluster.) Below each pair, we provide the URL of the mined Wikipedia article, followed by the cluster statistics. For each cluster, we report the number of photos, the number of users who took them, and the number of different days the photos were taken at. We also report the precision, obtained again by manual inspection as described above. In general the precision is very high, ranging between 93% and 99%. The precision values are also summarized in Figure 4.17.

Especially very well known landmarks, such as the Sacre Coeur (Figure 4.16 1), the Colosseum (4,5), or the Trevi fountain (14) are covered by a large number of photos with very few false positives. Lesser known objects, such as the Radcliffe Camera (15) have fewer images and are thus also more vulnerable to a few false positives. Staying with the Radcliffe Camera (15), note how multiple matching Wikipedia articles have been verified for the object. The same effect can be observed in example (13) or example (14), where articles in multiple languages were retrieved. Some matches are truly amazing, for instance example (5), where a painting matched to a photo of the Colosseum, or (12) and (13) with strong clutter and viewpoint change.

While most examples in Figure 4.16 refer to rather well known landmarks, some rare gems were mined, too. A few examples are shown in Figure 4.18. Example (1) does not only link to the article Sainte Chapelle, but also to an article about stained glass; similarly Mona Lisa (2) is linked to a specific article and a more general one about

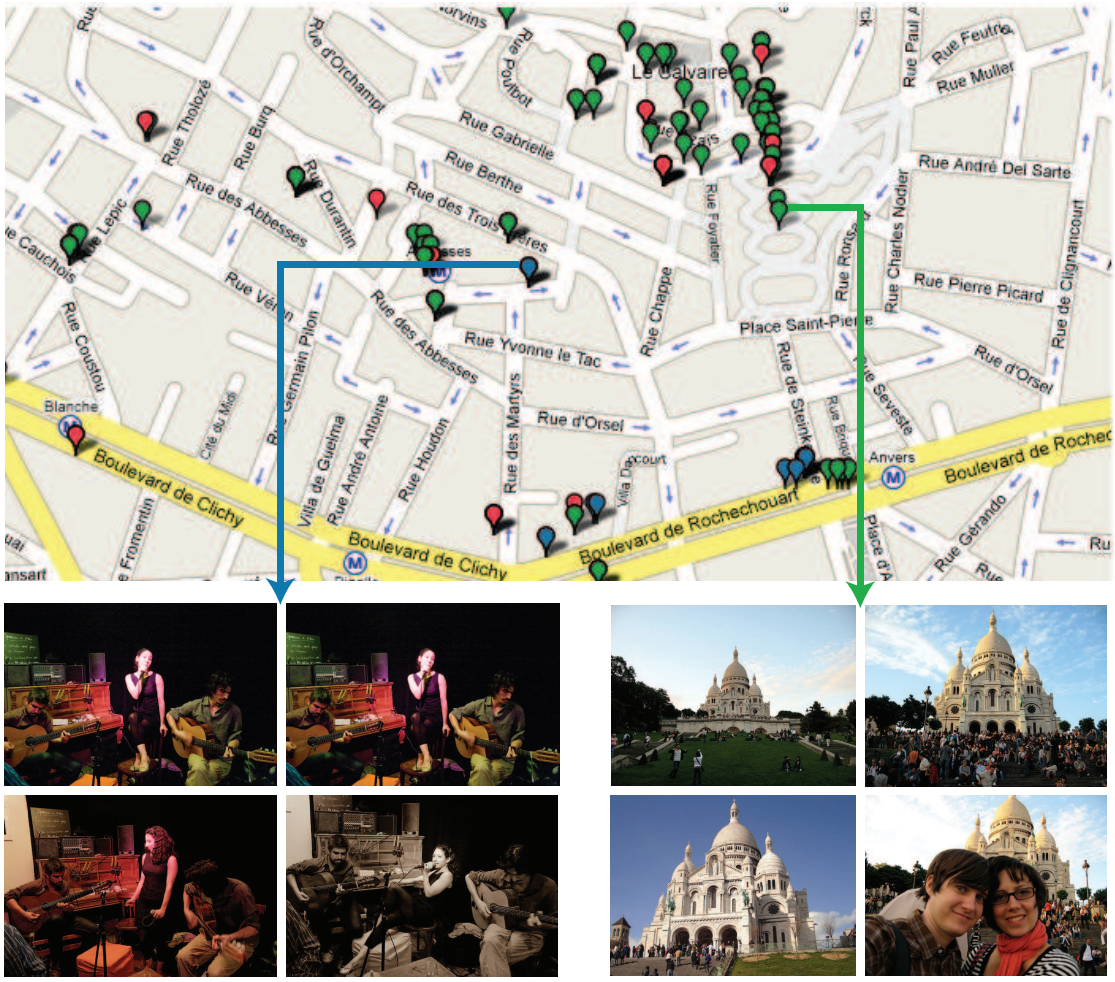


***Figure 4.15**: Object and event clusters on a map. Blue pins represent events, green pins objects. Red pins could not be classified as either one. Sample images of two selected clusters are shown below the map.*

Leonardo Da Vinci. In example (3), both the context "Forum Romanum" and the specific "Temple of Vesta" could be verified. Examples of smaller, even lesser known entities are shown in (4,5,6), note the maypole on Viktualienmarkt in Munich in (6): one of the articles explains the location, the other the tradition. Destinations with fewer tourists, such as Tallinn and Zurich (7,8) tend to have less photo coverage and also less content on Wikipedia. Nevertheless, some locations could be identified by our mining pipeline (7,8). Finally, example (9) is a lucky shot, where an event could be linked to a person and verified. By coincidence Wikipedia contains an image of an event (Jules Verne Adventures Film Festival, April 2007), which is also covered on Flickr and labeled with the attending actors' name. Clearly, only larger events are covered in Wikipedia, so that the chance of detecting a correct link for any event is rather small. Furthermore, homography based matching between images is well-suited for rigid objects and scenes, but less suited for events. Future work could

Figure 4.16: *A world tour with Flickr and Wikipedia. The left image in each pair stems from Wikipedia, the right image is the best match in a mined cluster. The Wikipedia links which could be verified this way are reported below the images, together with the cluster statistics. Note the high precision scores and the size of some clusters. (See text for a detailed discussion).*

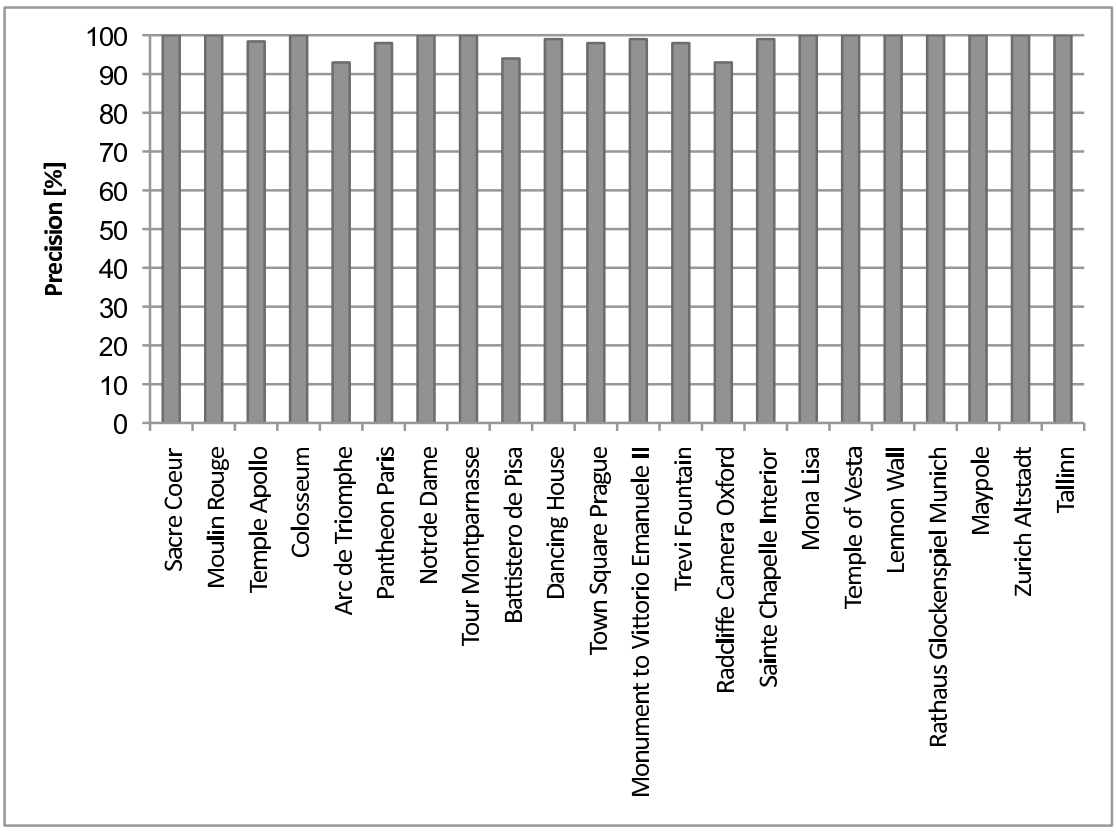


***Figure 4.17**: Precision within selected clusters.*

thus extend the system by classifying event scenes (wedding, concert, *etc.*) based on a bag-of-features approach [Bosch *et al.*, 2006] and label it using the textual meta-data rather than linking it to Wikipedia.

Table 4.4 contains statistics for the Wikipedia linking results. In total, 861 unique Wikipedia articles were verified by matching their images to our clusters as described above. The precision of this assignment was about 94%, *i.e.* 94% of the articles referred to a cluster which contained images of the article's correct subject. These articles covered 423 single-link clusters with 15′670 images. That is, about a quarter of all images in clusters could be related to a Wikipedia article.

Querying Wikipedia with the queries given by the frequent itemsets had resulted in over 20′000 URLs as linking candidates and in more than twice as many images parsed from the articles. This demonstrates how effective our method is in mining relevant links out of a vast amount of irrelevant data.

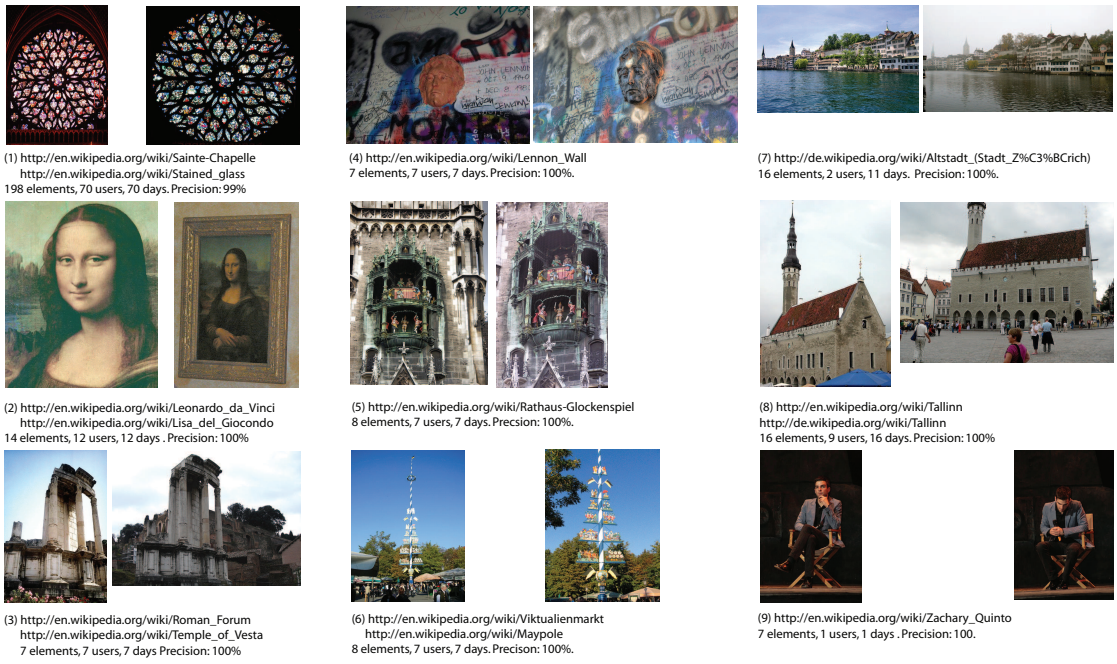


***Figure 4.18**: Additional, surprising mining results. See text for a discussion.*

4.6.4 Auto-annotation

Based on the object-specific feature confidence values derived in Section 4.5 we can estimate bounding boxes for mined objects, both for the existing database images but also for novel "query" images. We first show some results for object-level annotation of the mined database images. Combining the estimated bounding boxes and the information obtained from Wikipedia linking, we can create very appealing annotation displays. To that end, the links and tags are "attached" to the estimated bounding boxes. This is shown with a few examples in Figure 4.20.

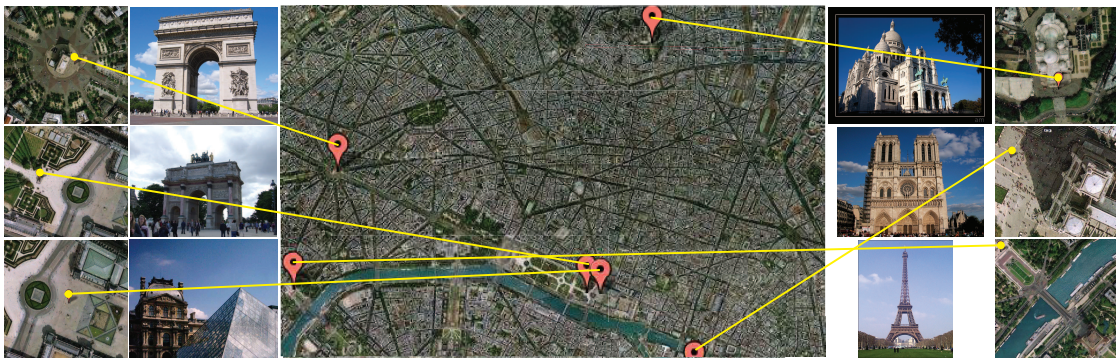


***Figure 4.19**: Auto-annotation of novel images using the mined clusters.*

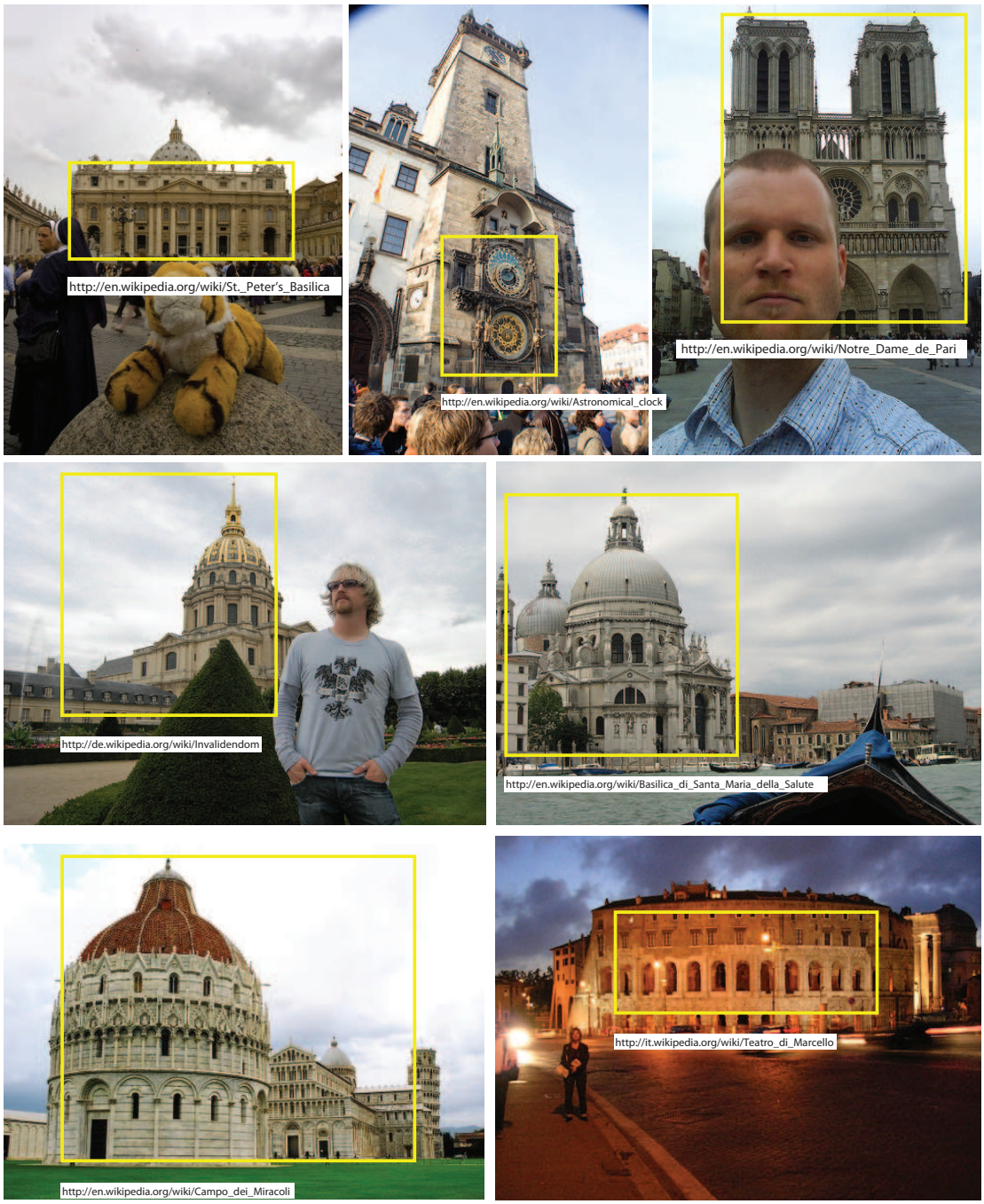


Figure 4.20: *Results of automatic object-level annotation with bounding boxes.*

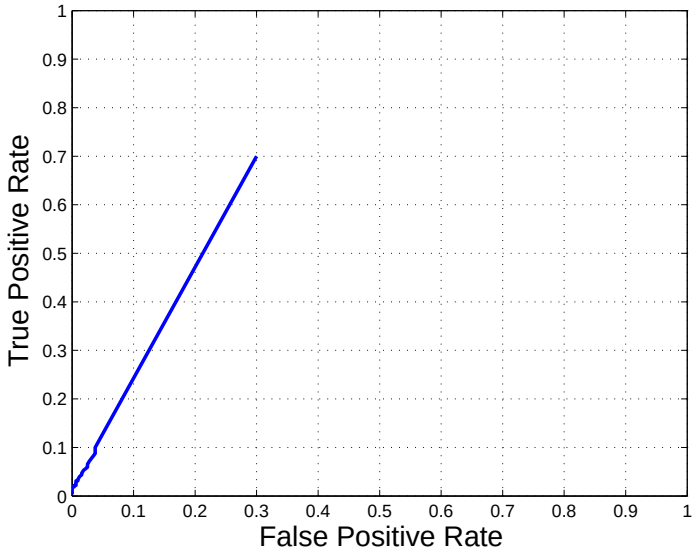


***Figure 4.21**: ROC curves for automatically created bounding boxes on mined data.*

To evaluate the quality of the object-level auto-annotation we created a groundtruth set of 320 images labeled with bounding boxes. More specifically, the groundtruth was created as follows: from the object-clusters that could be linked to a Wikipedia article, images were drawn at random. Each image and the corresponding Wikipedia URL were shown to an annotating person, who was given the task to label the object the Wikipedia article was referring to with a bounding box.

This annotated groundtruth was then compared to the bounding boxes detected by our system. A bounding box was counted as correct detection (true positive), if the intersection-over-union with the annotation was greater than 0.5. All other bounding boxes returned by our system were counted as false positives. This evaluation is the same that is commonly used in object class detection. Figure 4.21 shows ROC curves for that task. The curves were generated by varying through the range of the object-specific feature confidence thresholds the bounding boxes were estimated with. The overall recognition rate reaches 70%. This level is about the same as the values in retrieval tasks for specific objects such as [Philbin *et al.*, 2007]. However, the labelling relies purely on automatically detected cross-matches between clustered images, *i.e.* the system needs to decide automatically which fragments of a scene belong to the mined object and which fragments are part of the (sourrounding) background. Often, this discrimination is not easy, sometimes even hardly possible. Furthermore, sometimes the annotation refers to a larger scene, sometimes only to a specific object within a scene. Another source of error are clusters with few images which do not allow the calculation of a reliable feature confidence value, due to the lack of matching features. This is illustrated in Figure 4.22 (top), where the mean intersection over union (IOU) value for true positive detections is plotted over the

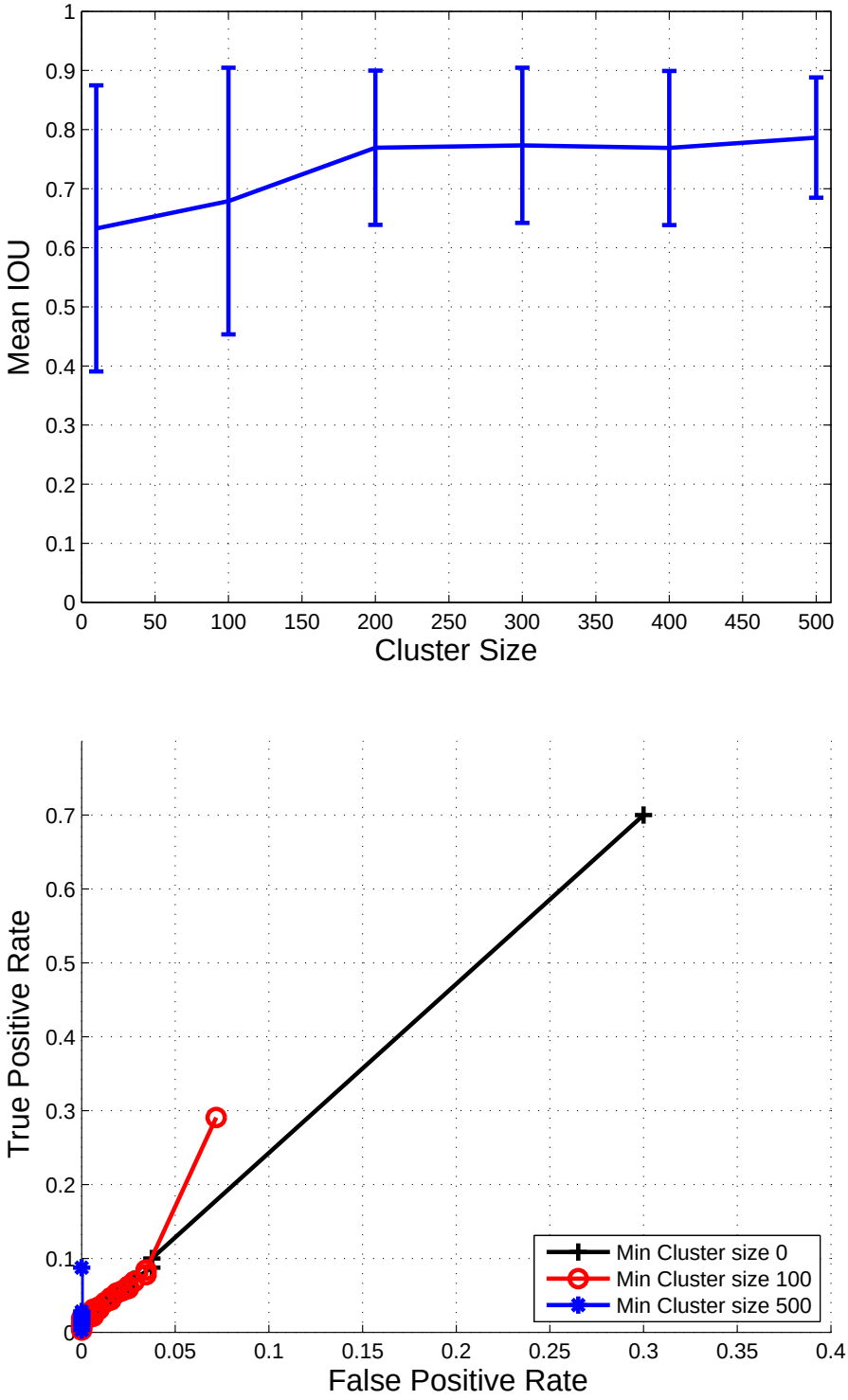


Figure 4.22: *Top: Mean intersection over union value (IOU) for detected bounding boxes at different minimal cluster sizes. Bottom: ROC curves by cluster size.*

cluster size for an object. Clearly, the IOU value is the higher the larger the cluster get. Figure 4.22 (bottom) shows corresponding ROC curves, *i.e.* when considering only detections "created" from clusters with the given minimal size. As expected, the ratio of true positive detections increases at the cost of recall the larger the minimal cluster size is chosen.

Selected true positive detection examples are shown in Figure 4.23, Figure 4.24 shows examples of false detections. It is worth noting, that most false positive detections are under-estimations of the true bounding-boxes. This is mostly due to lack of feature coverage. In total, 17′485 different photos could automatically be labeled with at least one bounding box. Note, that this refers only to photos which could also be linked to a Wikipedia article beforehand. With the database we built in this paper, annotation is not limited to the images that are already contained in the mined database. Auto-annotation of unlabeled "query" images with geo-location, object level bounding-box annotation, and corresponding Wikipedia article becomes also feasible. In a real world system, users could simply select the rough geographic area (*e.g.* by drawing a bounding box around Paris on the map), and photos would automatically be placed at their exact position and annotated with bounding boxes and Wikipedia articles. To demonstrate this capability, we downloaded 6 sample query images of sights in Paris from Google, see Figure 4.19. These are images which are neither present on Flickr, nor on Wikipedia. We load all clusters which we found in the Paris area (the full area as given in Table 4.2) and which could be assigned to a Wikipedia article, as described in the previous steps. These conditions hold for 167 clusters. Now, we simply match the query images to the clusters and record the best-matching image and cluster. This process only takes a few minutes, and the result is shown in Figure 4.19. The result location is selected as the mean location of all images in the matching cluster. Note the precision of the placement in the magnified map elements. All images are also linked to the correct Wikipedia article in the spirit of Figures 4.16 and 4.18. Note how similar the Arc de Triomphe and Arc de Triomphe du Carousel are (first and second image in the left column). Also note how close the two objects Arc de Triomphe du Carousel and the Louvre Pyramid are (second and third map in the left column). Our method is able to handle these uncertainties robustly and to discriminate between similar objects at different locations and different objects at the same location. In contrast, a direct matching of query images to Wikipedia images would not be possible in most cases, since the viewpoint changes might be too large. The number of images in our clusters (connected by a single-link clustering method) literally bridges the gap between the un-annotated query image and the Wikipedia image via the clusters created from Flickr data. Combining this method with scalable indexing [Philbin *et al.*, 2007] for local features will allow for auto-annotation of many holiday snaps within seconds.

Figure 4.23: *True positive detection examples. Yellow (dashed) line: annotation. Green (solid) line: detection.*

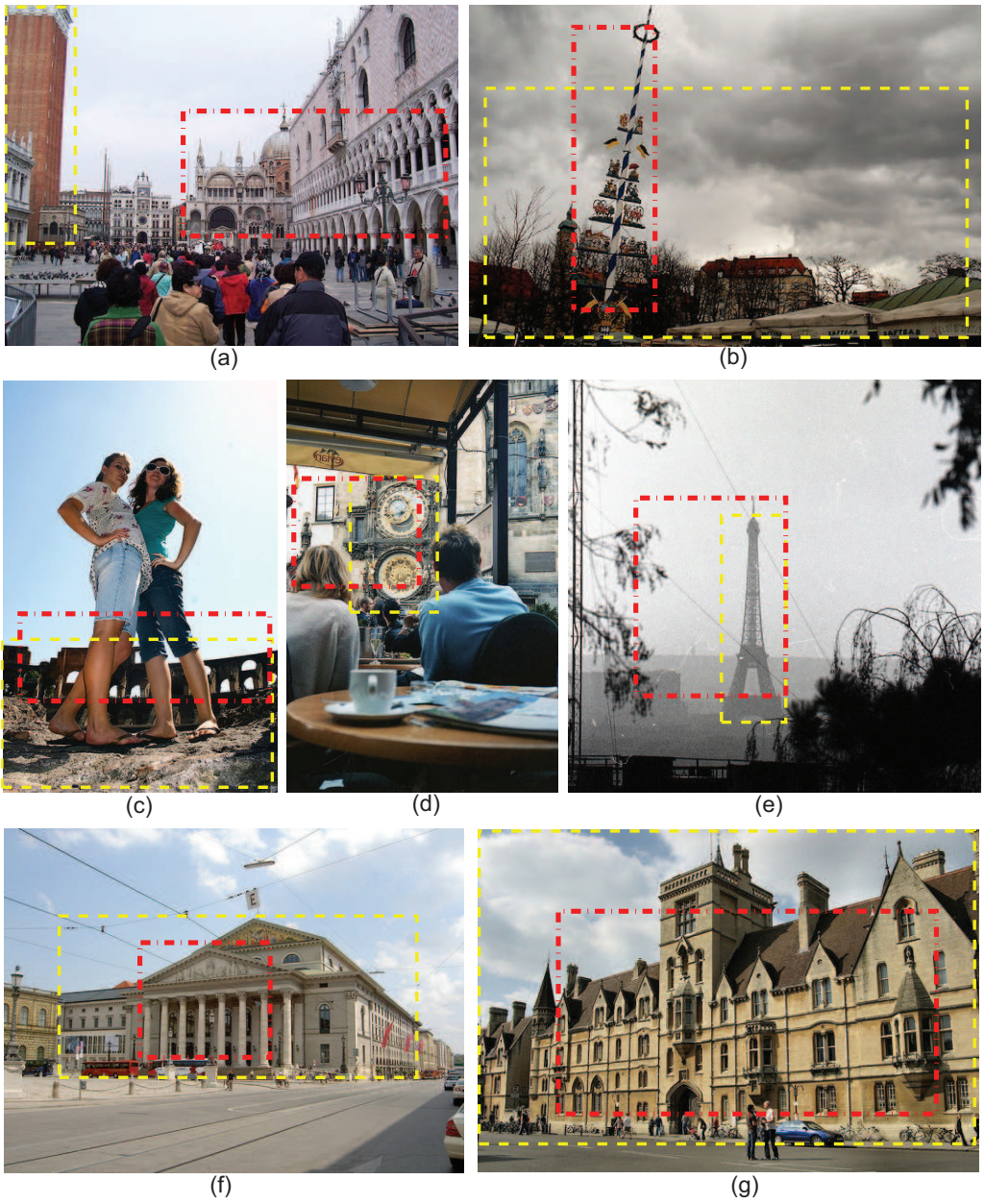


Figure 4.24: *False positive detection examples. Yellow line: annotation. Red line: detection. First row: False positives due to mismatch with annotation. In (a) the annotated Wikipedia article is 'St. Mark's Campanile' in (b) Viktualienmarkt. In both cases the learnt object-levlel annotation refers to another aspect at the same location. (St. Mark's Square and Maypole, respectively). Second row: particularly challenging examples. In (c) the groundtruth annotation for Colloseum includes the stone in the foreground, (d) extremely cluttered scene and small object (Prague Astronomical Clock), (e) difficulties of separating fore- and background, if images in cluster are taken from same viewpoint. (f) and (g): typical under-estimation of bounding box size due to lack of feature coverage.*

4.7 Related Work

Since the mining method proposed in this chapter covers an entire, multi-modal processing pipeline, it touches on a large variety of previous publications. Working with data from community photo collections has received increasing attention lately [Aurnhammer *et al.*, 2006; Jaffe *et al.*, 2006; Lew *et al.*, 2006]. However, most of those approaches are based either on text [Jaffe *et al.*, 2006] or only global visual features. The local visual features which are used in this work, however, allow to find very good and extremely accurate matches between the depicted objects even under significant changes in viewpoint, imaging conditions, scale, lighting, clutter, noise, and partial occlusion. A similar approach would not be possible using global measures such as color or texture histograms. Philbin and Zisserman [Philbin *et al.*, 2007] also worked with local features and multiple view geometry on a database of landmark buildings obtained from Flickr. The main goal of that work was to derive a scalable indexing method for local visual features, the database was retrieved and annotated manually. The work most similar to ours is probably [Simon *et al.*, 2007; Snavely *et al.*, 2006]. Here, the authors also proposed clustering images from community photo collections using multi-view geometry based matching between images. The goal was to derive canonical views for certain landmarks and to use those as entry points for browsing. Initial image collections were retrieved by querying photo collections with known keywords such as "Rome", "Pantheon", *etc.* As we will demonstrate, our fully unsupervised approach based on geographic tiling is not only more flexible, but also more scalable. (The dataset used in [Simon *et al.*, 2007] contained 20′000 photos, while ours is one order of magnitude larger). Furthermore, we add several layers of processing which extract semantic information, such as classification into objects and events, and which automatically include other content sources such as Wikipedia for unsupervised labeling of objects. To the best of our knowledge, this work is the first to propose this kind of pipeline, taking as an input only a geographic tiling of the world and resulting in an output of automatically mined landmark objects, together with their semantics in the form of automatically created links to Wikipedia.

Similar in scale are the experiments conducted in the recent work [Hays and Efros, 2008]. Here, the geographic location an image was taken at is estimated by comparing it to a huge database of images downloaded from Flickr. The overall objective is to find near duplicate images of the same scene very efficiently. To that end, the authors process 6 million geo-referenced images to create a reference database with good coverage of the earth. The images are encoded using several global feature types (tiny images [Torralba *et al.*, 2008], GIST [Oliva and Torralba, 2001], color histograms, *etc.*). Estimating the location the picture was taken at is now simply done by finding the nearest neighbour(s) in the database. This works astonishingly well, reaching absolut recognition rates of up to 16% for locating an unseen test-

image within $200km$ of its correct location. However, these recognition rates are not sufficient for the applications we have in mind. The method also don't allow precise description of objects in the images, as with our system, which builds on local features instead of global ones. In summary, [Hays and Efros, 2008] is a work quite complementary to ours.

Finally another approach with the focus on reconstructing 3D models from images collected in community photo collections similar to [Simon *et al.*, 2007], was just recently proposed in [Li *et al.*, 2008b].

4.8 Discussion and Conclusions

We have presented a fully unsupervised mining pipeline for community photo collections. The sole input is a grid of tiles on a world map. The output is a database of mined objects and events, many of them labeled with an automatically created and verified link to Wikipedia. The pipeline chains processing steps of several modalities in a highly effective way. The basis is a pairwise similarity calculation with local visual features and homography-based geometric verification for each tile. Hierarchical clustering was demonstrated to be a very effective method to extract clusters of the same entities in different contexts (indoor, outdoor, *etc.*). We observed that the clustering step on visual data is far more reliable than on text labels. A simple tree-based classifier on the metadata of photos was introduced to discriminate between object an event clusters. Itemset mining on the text of the clusters created with visual features was proposed to mine frequent word combinations per cluster. Those were used to search Wikipedia for potentially relevant articles. The relevance was verified by matching images from the Wikipedia articles back to the mined clusters. Both the clustering and linking to Wikipedia showed high precision. Finally, in a last step we demonstrated how the database can be used to auto-annotate unlabeled images without geotags down to object-level annotation of objects with bounding boxes, assignment of geographi location, textual tags and related content.

Besides the effective mining pipeline proposed in the paper, we also carried out one of the largest experiments with local visual features on data from community photo collections by processing over $200'000$ photos. The results of this large-scale experiment are very encouraging and open a wealth of novel research opportunities. They include in particular improved auto-annotation of data from multimodal information sources, processing at even larger scales by integrating scalabale feature matching methods and distributed processing, and more precise object level annotations. Combinations with complementary works such as [Hays and Efros, 2008; Simon *et al.*, 2007; Snavely *et al.*, 2006] would allow for interesting applications. Finally, integrating mining methods and scalable retrieval, in combination with con-

tinually growing amounts of available data will probably lead to the creation of very exciting auto-annotation and retrieval systems in the coming years.

5

Retrieval in a Multimodal Context

In the preceding chapters we looked into mining visual data at several levels. We mined basic feature configurations in Chapter 3, and in Chapter 4 we proposed a system which automatically mines objects and events from community photo collections. In this chapter, we access the databases from the retrieval side. Stepping one more step closer to a fully functional object recognition system, we look into the retrieval process and the user-interaction with the system, including the user interface.

Retrieval and mining are closely related topics. Mining data from sources such as the Web, allows us to create indices and entry points to access the data. If we are able to detect occurrences of certain object classes (Chapter 3), the database could be accessed by searching for the corresponding class names. Mining specific objects and their descriptions from the Internet (Chapter 4) allows us to access the index either with text search to find images depicting the object, or with a query image of the object, to learn more about it, for instance by reading a Wikipedia article. Such a capability has a wealth of applications, especially for the rising number of camera-equipped mobile phones in use, and the growing amount of digital imagery being shared on the Web.

The Web and the use of mobile devices for retrieval define a multimodal context, which should be exploited by retrieval applications. We exploited the multi-modality of the data in community photo collections in the previous chapter to make the connection between an object and related Internet content. This allows retrieval with one modality, requesting content for another as result. For instance, taking a picture of a building with a mobile phone returns related Wikipedia pages. The multimodal context can also be used to support the retrieval process. For example, the geographic position of the user making a request from a mobile device can be used to restrict the search space to objects within her vicinity.

This chapter explores such applications by discussing several prototype applications that were implemented in the course of this thesis. The chapter is organized as follows. In Section 5.1 we look shortly into the history of image retrieval. In Section 5.2

we discuss retrieval applications for mobile devices, and the potential impact these devices will have as an interface in the near future. This is followed by an evaluation of several prototype applications for mobile devices, including a city-guide and a slide-tagging application for meeting rooms. In Section 5.3 we focus on applications for the Web. Finally, in Section 5.4 we propose an approach to locate and extract text in natural scenes, as an additional means to support multimodal retrieval in visual data. The chapter concludes with a discussion of related works and a summary of the results.

5.1 The Query by Example Paradigm Revisited

Understanding and interpreting the content of an image is one of the fundamental problems of computer vision research. A substantial amount of work has been carried out in the 1990's, trying to exploit global feature descriptors such as color, shape and texture to build image retrieval systems. The typical usage scenario proposed to start with an example image as a query and to use its (global) features to retrieve similar images from a database. Similarity is typically expressed as a distance in feature space either for a single feature type or a combination of multiple types. Such an interaction with the system is commonly termed Query by Example (QBE). Some of the early systems that worked according to this paradigm include the QBIC project [Flickner *et al.*, 1995] at IBM, BlobWorld [Carson *et al.*, 1999], NeTra [Ma and Manjunath, 1999], PicHunter [Cox *et al.*, 2000] and MARS [Rui and Huang, 1999], and many others.

However, relying on an example image as a starting point for the search also arrives with certain limitations to the system's usefulness. After all, where would the user get the example image from? And if he had it already, why would he want to search for another image with similar content? QBE is typically combined with other entry points into the retrieval process. One approach consists of initiating search with a text retrieval on the meta-data associated with the images in the database to obtain a set of initial sample images. The process then continues with content-based image retrieval by selecting the sample closest to the expected result. Cortina [Quack *et al.*, 2004] was one of the systems which applied this solution to search a few million images from the Web.

Other approaches are category-based browsing, or retrieval within the database, *i.e.* the queries are selected from (sub-)images in the database. The latter is especially relevant for video data, where the task consists of finding an object from a query frame in other shots of the video. Here, the influential work by Sivic and Zisserman [Sivic and Zisserman, 2003] probably marks the turning point, where interest in the community shifted from similarity retrieval to object-level retrieval and from global image features to (visual vocabularies of) local appearance features.

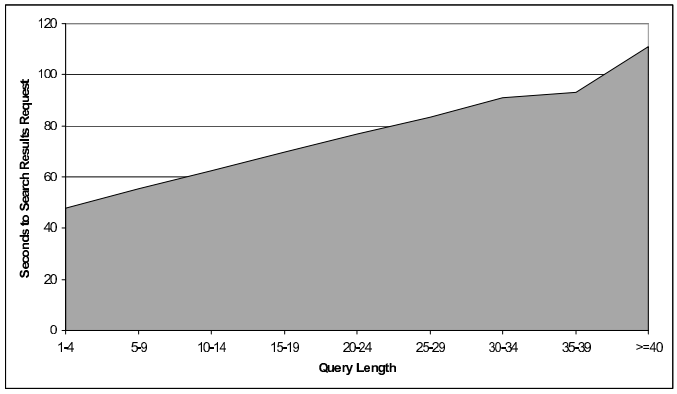


***Figure 5.1**: Time required to enter a keyword query on a mobile device in relation to query length (in characters). Figure by [Kamvar and Baluja, 2006]*

Similarity retrieval and retrieval within a closed database are of interest mostly to users in the publishing and media industry, trying to find similar data for a given image, for instance when looking for illustrations and photos to illustrate news articles, or archive video data for a given topic.

With the widespread availability of digital cameras, on-line photo sharing platforms, and mobile phones with integrated cameras, the QBE concept in its "purest" form is suddenly of relevance again: using a mobile phone, the user can "generate" an example-image with a single click and even transmit it to a retrieval system automatically. With this application the focus shifts to identifying a specific object in the query image, rather than retrieving similar images. In other words, the main purpose of the system is not to return more images of the same, but to identify a specific object in the image and to return multimodal information about it. This could ease Web search from mobile devices: instead of tedious typing of keyword queries on small buttons, sending a picture is sufficient to start searching. Figure 5.1 shows the results from a study [Kamvar and Baluja, 2006] conducted by Google. It shows the time a user spends typing a query on a mobile phone keypad for a given number of characters in the query. It is quite stumbling, that the average time spent typing amounts to roughly 40s. Beyond the limitations imposed by typing of queries, in some cases the user may not even know the correct query to enter, *e.g.* for an unknown landmark building in a foreign city. This concept, *i.e.* the possibility to interact efficiently with physical objects (or "things" for that matter) and to access digital information about them is often termed "The Internet of Things".

The following section explores the application of object recognition in this context. We put an emphasis on the user perspective on image and video retrieval by investigating various applications, systems and user interfaces for object-level retrieval in visual databases.

5.2 Object Recognition for Mobile Devices

Extending the Internet to physical objects — the Internet of Things — promises humans to live in a smart, highly networked world, which allows for a wide range of interactions with this environment. One of the most convenient interactions is the request for information about physical objects. For this purpose several methods are currently being discussed. Most of them rely on some kind of unique marker integrated in or attached to the object. Some of these markers can be analyzed using different kinds of wireless near field communication (for instance RFID tags [Want, 2004] or Bluetooth beacons [Fuhrmann and Harbaum, 2003]), others are visual markers and can be analyzed using cameras, for instance standard 1D-barcodes [Adelmann *et al.*, 2006] or their modern counterparts, the 2D codes [Rohs and Gfeller, 2004].

A second development concerns the input devices for interaction with physical objects. In recent years mobile phones have become sophisticated multimedia computers that can be used as flexible interaction devices with the user's environment. Besides the obvious telephone capabilities, current devices offer integrated cameras and a wide range of additional communication channels such as Bluetooth, WLAN and GPRS/UMTS/3G access to the Internet. People are used to the device they own and usually carry it with them all day. Furthermore, with the phone-number, a device is already tied to a specific person. Thus it is only natural to use the mobile phone as a personal input device for the Internet of Things.

Indeed, some of the technologies mentioned above have already been integrated in mobile phones, for instance barcode readers or RFID readers. The ultimate system, however, would not rely on markers to recognize objects, but rather identify it by their looks, *i.e.* using visual object recognition from a mobile phone's camera image. Since the large majority of mobile phones contain an integrated camera, a significant user base can be addressed at once. With such a system, snapping a picture of an object would be sufficient to request all the desired information on it. While this vision is far from being reality for arbitrary types of objects, with the methods presented in the preceding chapters we are able to recognize certain types of objects very reliably and "hyperlink" them to digital information.

Using object recognition methods to hyperlink physical objects with the digital world brings several advantages. For instance, certain types of objects are not well suited to attach markers. This includes also large landmark buildings, where markers might only be attached at few locations at the building. (such an experiment has been attempted with the Semapedia project [1]). Furthermore, a user might want to request information from a distance, for instance for a church tower which is up to several hundred meters away. But even if the object is close, markers can be impractical. A

[1] http://www.semapedia.org

barcode or RFID attached to the label of an object displayed in the museum would be difficult to access if the room is very crowded. Taking a picture of the item can be done from any position where it is visible. Furthermore, consistent tagging of the objects is often difficult to achieve. One example are outdoor advertising posters. If a poster company wanted to "hyperlink" all their poster locations, they would have to install an RFID or bluetooth beacon in each advertising panel or attach a barcode to each of them, which requires a standardized system and results in costs for installation and maintenance. Another field of application are presentation screens in smart meeting rooms or information screens in public areas. The content displayed on the screen is constantly changing and it would be an involved process to add markers to all displayed content.

Using object recognition to interact with these objects requires only a database of images. That being said, object recognition does not come without restrictions, either. For instance, it is currently (and maybe always) impossible to discriminate between highly similar objects, such as two slightly different versions of the same product in a store. Furthermore, efficient indexing and searching visual features for millions or billions of items is still a considerable research challenge. (Chapter 6 of this thesis presents some possible methods to scale retrieval in databases of local image features).

5.2.1 Mobile Interfaces

In this section we discuss some of the options for system architecture and user interaction when designing a retrieval system for mobile devices. The options to consider include the distribution of processing tasks to client or server side, or wether a request-response or a real-time streaming interface should be offered to the user. Many of these options depend on the processing capabilities of today's mobile phones and on the bandwidth available on mobile communication networks. In the following sections we discuss some prototype applications of mobile user interfaces we implemented.

Client-side vs. Server-side Processing

One of the first questions to consider is which tasks can or should be done on the phone itself and which tasks are better delegated to a server-side processing system. One extreme would be to implement a whole system including feature extraction, database storage and database search on the mobile phone itself. This is currently only possible for applications with small databases (a few thousand items at most, probably) and on high-end devices due to limitations of CPU and memory. An overview of the capabilities of currently available phones is given in Table 5.1. It

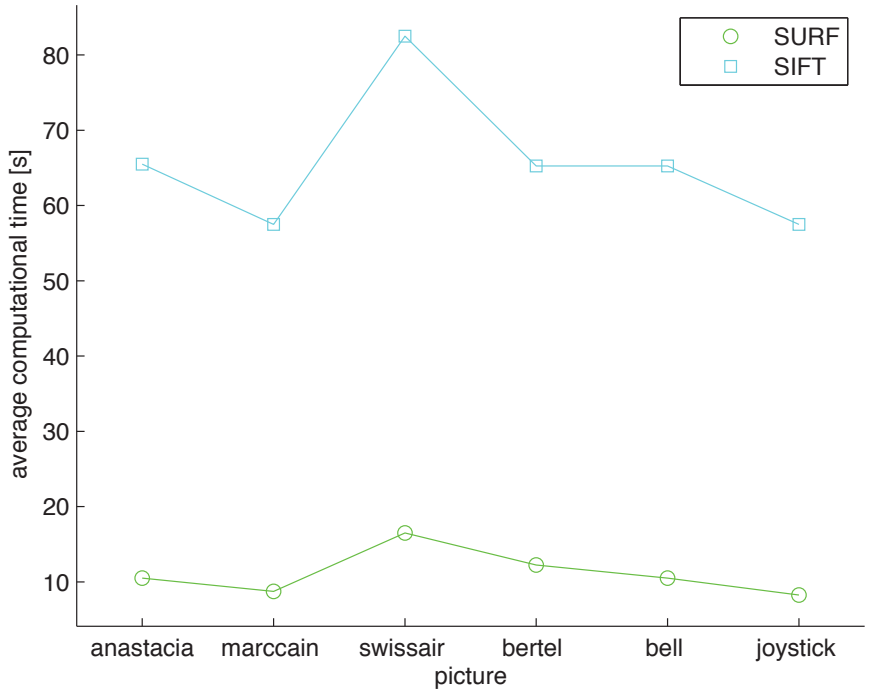


***Figure 5.2**: SIFT and SURF on a mobile Phone (Nokia 6630).*

can be seen that CPU speed is a few hundred MHz at most, and only the latest and most expensive devices have more than 100MB of RAM.

Phone Model	Type	Year	CPU	RAM	GPS	WiFi	Camera
Nokia 6230	C	2004	?	6 MB	no	no	0.3 MP
Nokia 6630	HE	2004	220 Mhz	10 MB	no	no	1.3 MP
Nokia N70	HE	2005	220 MHz	22 MB	no	no	2.0 MP
Nokia N95	HE	2007	332 MHz	128 MB	yes	yes	5.0 MP
Apple iPhone 3G	HE	2008	412 MHz	128 MB	yes	yes	2.0 MP

***Table 5.1**: Capabilities of typical mobile phones. Type denoted as: HE (High-end device). C (Simple consumer device).*

For larger databases, one could consider moving at least the feature extraction to the client, and sending features as query to a server. While the set of local feature descriptors for a typical query image is usually not more compact than a compressed image, the advantages would be distributed processing for the feature extraction, and increased privacy, since only features instead of images are transmitted. We thus implemented SIFT and SURF feature extraction on the Symbian [Edwards, 2004] mobile platform using platform specific C++.

Feature extraction runtimes using SIFT and SURF for a few typical images are shown in Figure 5.2. The implementation is a rather straightforward port of the workstation source codes, *i.e.* not optimized for the mobile beyond the changes that are required due to the architecture of the embedded platform. It can be seen, how SURF outperforms SIFT by about the same factor as on a PC. However, absolute runtimes are extremely high, in average more than $10s$ for SURF on a typical image. In contrast, on a modern PC SURF feature extraction takes a few hundred ms [Bay *et al.*, 2006b]. The absolute recognition times could be reduced by avoiding floating point operations. Only just recently client-side applications have been proposed [Wagner *et al.*, 2008] which allow real-time extraction of SIFT features on client-side devices through heavy optimization.

Considering the fragmentation of the mobile phone ecosystem (*e.g.* operating systems, processors, *etc.*) and the rather low processing capabilities of even high-end devices, a server-side approach for object recognition seems preferable. The main challenge is now posed by transmitting the image data to the server and rendering the response. Here, we implemented three prototype applications:

1. Single shot server-side processing with manual release
2. Continuous real-time recognition from video streams
3. Hybrid: Single shot server-side processing with automatic release

The implementation on the devices was carried out in several student projects, detailed descriptions can be found in the respective reports [Breu and Müller, 2008; Jecker and Knecht, 2008; Ulrich, 2006].

Single shot server-side processing with manual release

The simplest application consists of sending a single image initiated when the user presses a button on the phone. The image is transmitted to a server, sent through an object recognition pipeline, and the response is sent back to the phone. Often, the response will be an URL to a web-page with information about the recognized object. Thus, our sample application opens the phone's internal browser and renders the web-page for the URL. This process is shown in Figure 5.3 with screenshots from our application. The client-side software was programmed in C++ for Symbian.

Continuous real-time recognition from video streams

A more user-friendly application than the one in the previous section would label recognized objects continuously on the phone's screen. In a server-side implementation of the actual object recognition pipeline, this requires sending a continuous

***Figure 5.3**: Client software for the cityguide application: the user snaps a picture, waits a few seconds, and is redirected to the corresponding Wikipedia page.*

video stream to the server and detecting objects from it. Our implementation [Jecker and Knecht, 2008] builds on available open-source packages [2] to allow MPEG video-streaming from Symbian devices via Bluetooth or Wireless LAN to a server. We extended the server software with a thread for object recognition. Incoming frames are matched to the database of objects at regular time-intervals. If a match is detected, the object is tracked through the subsequent frames. Tracking is done by simply matching local features of the database object continuously (SURF features allow this kind of real-time matching) and the coordinates of a bounding box around the matched features are sent back to the client. Along with the bounding box, a string with the title is transmitted. The Symbian client was extended to receive both the bounding box and the string with the object's name, and display them on the screen accordingly. This is shown in Figure 5.4.

When the object is lost while tracking, the database is queried again with the incoming frames, and the process above is repeated. Note, that the system is currently limited to recognize on object in the field of view, but could easily be extended to handle multiple recognitions. Obviously, receiving and processing many parallel streams would put a lot of burden on a server system, too.

[2] http://www.movino.org/

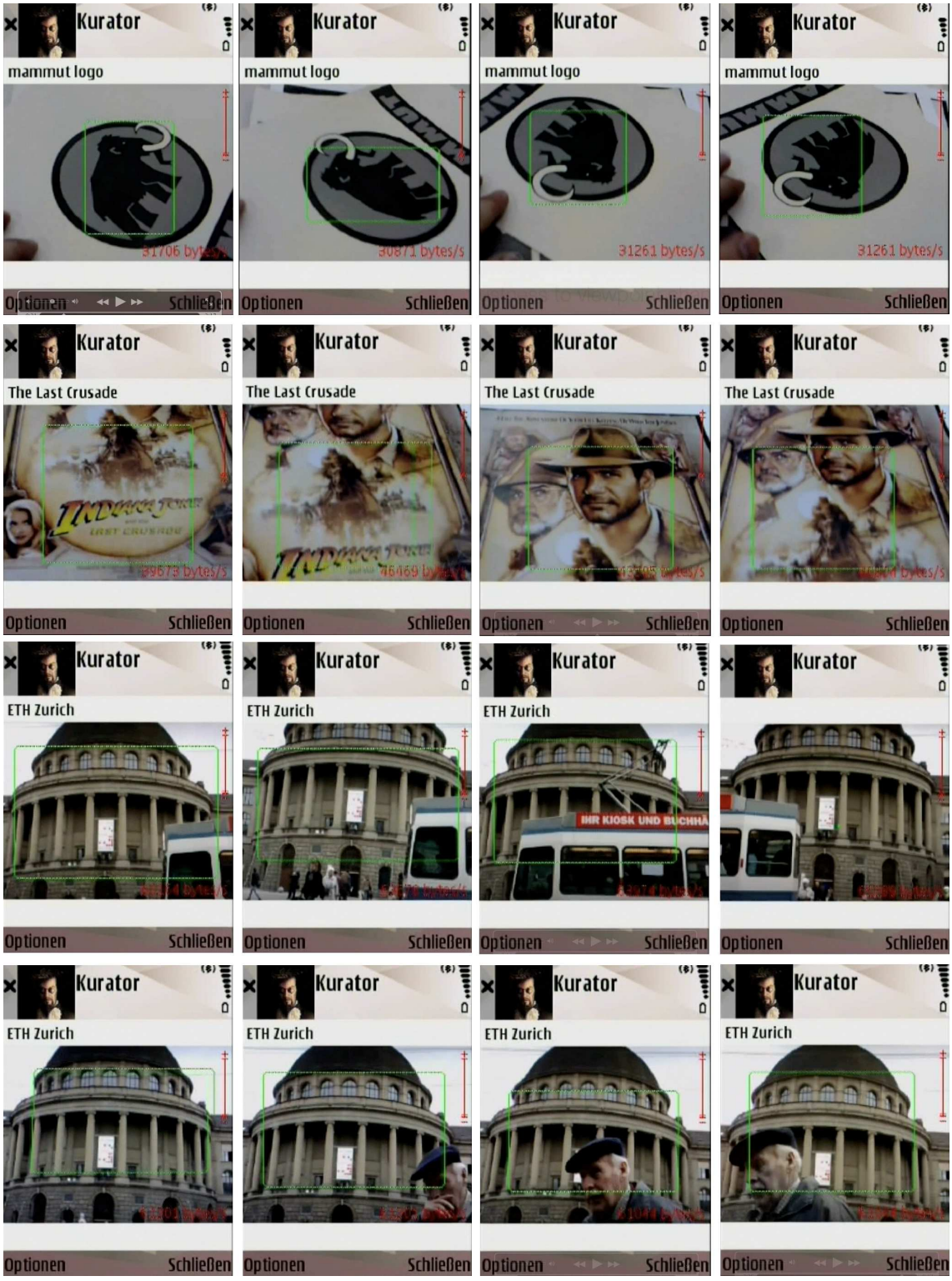


***Figure 5.4**: Screenshots of our real-time, server side object recognition system for mobile devices.*

Hybrid: Single shot server-side processing with automatic release

Finally, we implemented an intermediate or hybrid approach. The idea is to lower the burden on the server by avoiding processing of live video streams, but maintaining usability over the single shot version. We propose to initiate queries from the client automatically, when appropriate. More specifically, when the user holds the camera still (pointed at a target), a request will be initiated. The appropriate time to initiate a request will be determined by analyzing motion on the client device. We implemented a prototype based on optical motion detection from the camera video feed on the client device. For that purpose we relied on an implementation of motion history images [Davis, 2001], which are part of the Nokia Computer Vision Library for Symbian[3]. If the observed motion falls below a predefined threshold, a request is sent to the server and the name of the detected object (if any) is displayed on the screen. Figure 5.5 shows an example.

This approach could be extended by implementing client-side tracking of objects. The tracking algorithm on the phone should be sufficiently simple to run in real-time under the restrictions posed by the processing capabilities by todays mobile phones. To handle drift, tracking could be verified by sending a request to the server for more precise positioning and re-initialization of tracking on the client.

5.2.2 Sample Applications

In this section we propose and evaluate two sample application scenarios for object retrieval from camera-equipped mobile phones.

The first one is slide tagging in smart meeting rooms. Users have the ability to "click" on slides or sections of slides that are being presented to record them for their notes or to add tags. The second application is a cityguide on the mobile phone. Users have the possibility to take a picture of a sight, send it to a recognition service, and receive the corresponding Wikipedia article as an answer. For this application, the search space is limited by integrating location information, namely cell-tower ids or GPS.

Both systems are experimentally evaluated in different dimensions, including different phone models with different camera qualities, for the trade-offs using different kinds of search space restriction (geographic location etc.), and with and without projective geometry verification stage.

[3] `http://research.nokia.com/research/projects/nokiacv/`

***Figure 5.5**: Motion detection for a mobile visual search interface.*

5.2.3 Hyperlinked Slides: Interactive Meeting Rooms

Today's meeting rooms are being equipped with an increasing number of electronic capturing devices, which allow recording of meetings across modalities [Abowd, 1999; Amir *et al.*, 2001]. They often include audio recording, video recording, whiteboard capturing and, last but not least, framegrabbing from the slide projector. These installations are usually deployed to facilitate two tasks: allowing off-line retrieval and browsing in the recorded meeting corpus and turning the meeting rooms into smart interactive environments. In the work at hand, we focus on the captured presentation slides which are a central part of today's presentations. As shown in Figure 5.6, the slides usually contain the speaker's main statements in written form, accompanied by illustrations and pictures, which facilitate understanding and memorizing the presentation. Indeed, the slides can be seen as the "glue" between all the recorded modalities. Thus, they make a natural entry point to a database of recorded presentations.

A typical usage scenario for such a system could look as follows: Using the integrated camera of her mobile phone, an attendee to a meeting takes a picture of a slide which is of interest to her. The picture is transmitted to a recognition server over a mobile Internet connection (UMTS, GPRS *etc.*). On the server, features are extracted from the picture and are matched to the database of captured slides. The correct

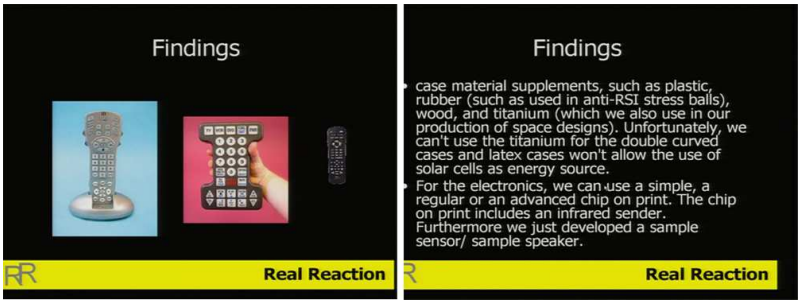


Figure 5.6: *Typical presentation slides from the AMI corpus database.*

Figure 5.7: *The user "tags" a presented slide using our mobile application by taking a picture (left), which is automatically transmitted to the server and recognized (middle), a response is given in an automatically opened WAP browser (right).*

slide is recognized, added to the users' personal "bookmarks", and she receives a confirmation in a WAP browser on her mobile phone. Note that the messaging from the phone can be done using standard MMS or using a custom client-side application which we programmed in C++ on the Symbian platform. Figure 5.7 shows screenshots of our mobile application for a typical usage scenario.

Back at her PC, the user has access to all her bookmarked slides at any time, using a web frontend which allows easy browsing of the slides she bookmarked. From each bookmarked slide she has the possibility to open a meeting browser which plays the other modalities, such as video and audio recordings, starting at the pint in time the slide was displayed. By photographing only a section of a slide, the user has also the possibility to highlight certain elements (both text or figures) — in other words, the mobile phone becomes a digital marker tool.

Of course one could assume a very simple slide bookmarking method, which only relies on timestamping. The client-side would simply transmit the current time, which would be synchronized with the timestamped slides. Our system does not only allow for more flexible applications (the aforementioned "highlighting" of slide elements) but is also more robust against synchronization errors in time. In fact, using a "soft" time restriction of some minutes up to even several hours would make our system more scalable and unite the best of both worlds. Finally, a system like ours could also discriminate between multiple parallel sessions, which are common at larger conferences.

The basic functionality of the proposed slide recognition system on the server is as follows: for incoming queries, scale invariant local features are extracted. For each feature a nearest neighbor search in the reference database of slides is executed. The resulting putative matches are verified using projective geometry constraints. The next two subsections describe these steps in more detail.

Slide Recognition System

We start from a collection of presentation slides which are stored as images. This output can be easily obtained using a screen capture mechanism connected to the presentation beamer. From the image files, we extract scale invariant features around localized interest points. In our implementation we use again SURF [Bay *et al.*, 2006b] detector and descriptor combination.

Slide recognition consists again of the two steps feature matching and global geometric verification. For the feature matching we compare the feature vectors from the query image to those of the images in the database. In this example we use linear feature matching based on the Euclidean distance as in the previous chapters. Since the database objects (the slides) are planar, we can again rely on a 2D homography

mapping [Hartley and Zisserman, 2004] for the geometry filter. The result of such a geometric verification with a homography is shown in Figure 5.8.

Experiments

For our experiments we used data from the AMI meeting room corpus [Carletta et al. (17 authors), 2005]. This set contains the images of slides which have been collected over an extended period using a screen-capture card in a PC connected to the beamer in the presentation room. Slides are captured at regular time intervals and stored as JPEG files. To be able to synchronize with the other modalities (*e.g.* speech and video recordings), each captured slide is timestamped.

To create the ground truth data, we projected the slides obtained from the AMI corpus in our own meeting room setting and took pictures with the integrated camera of two different mobile phone models. Namely, we used a Nokia N70, which is a high-end model with a 2 megapixel camera, and a Nokia 6230, which is an older model with a low quality VGA camera. (See Table 5.1 for a detailed comparison of the phone models.) We took 61 pictures with the N70 and 44 images with the Nokia 6230 [4]. Figure 5.9 shows some examples of query images. The reference database consists of the AMI corpus subset for the IDIAP scenario meetings, which contains 1098 captured slide images.

We extracted SURF features from the reference slides in the database at two resolutions, 800x600 pixels and 640x480 pixels. For the 1098 slides this resulted in a database of $1.02 * 10^6$ and $0.72 * 10^6$ feature vectors, respectively. For the SURF feature extraction we used the standard settings of the detector.

The resolutions of the query images were left unchanged as received from the mobile phone camera. We ran experiments with and without homography check, and the query images were matched to the database images at both resolutions. A homography was only calculated if at least 10 features matched between two slides. If there were less matches or if no consistent homography model could be found with RANSAC, the pair was declared unmatched. If there were multiple matching slides, only the best was used to evaluate precision. Since the corpus contains some duplicate slides, a true match was declared if at least one of the duplicates was recognized.

Table 5.2 shows the recognition rates, for the different phone models, different resolutions and with and without homography filter. At 800x600 resolution, the homography filter gives an improvement of about 2% or 4% for each phone type, respectively. The recognition rate with a modern phone reaches 100%, the lower quality camera in

[4]The query images with groundtruth are made available for download under http://www.vision.ee.ethz.ch/datasets/.

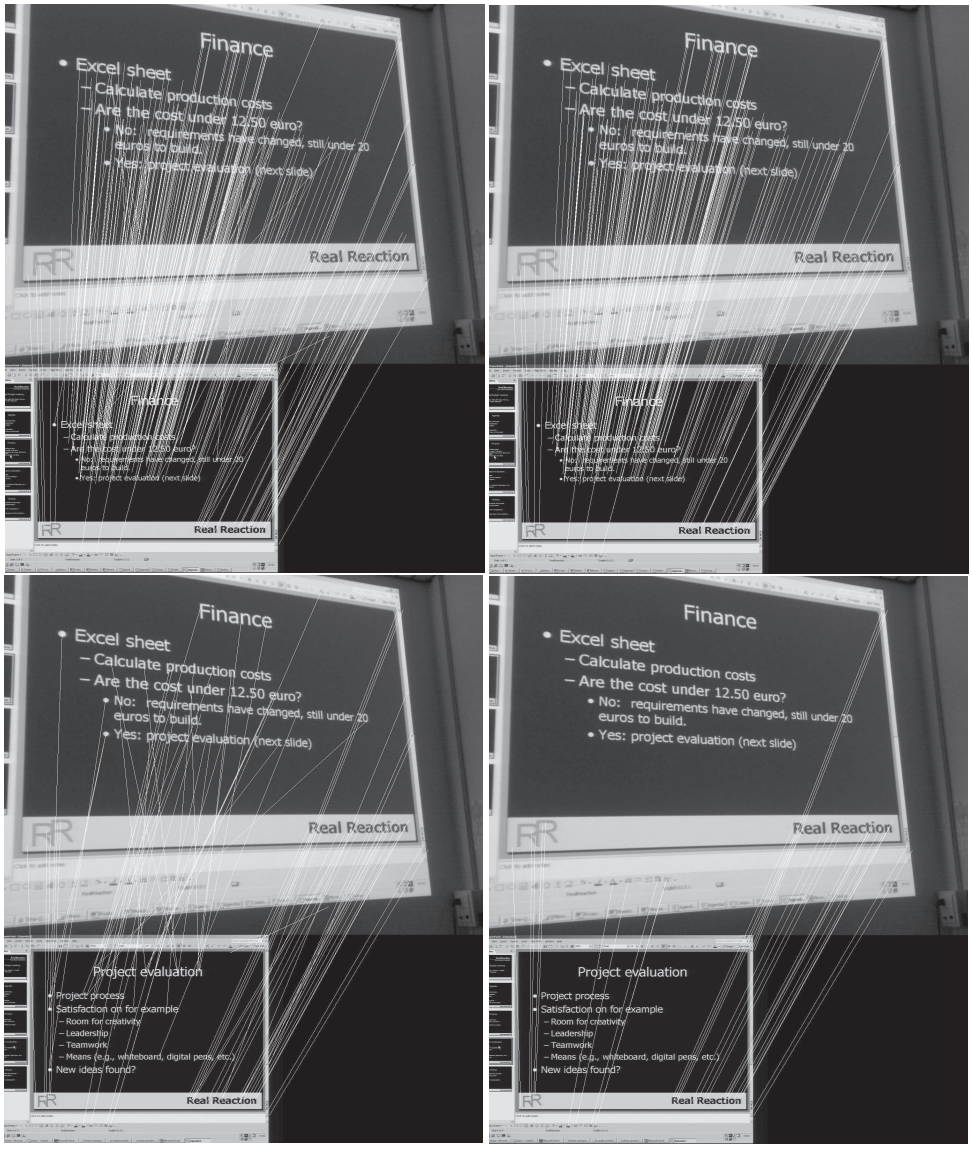


Figure 5.8: *Geometric verification with a homography. Top rows: matches for a query image with the correct database image. Top left: before homography filter, top right: after homography filter. As the match between the slides is correct most of the putative feature matches survive the homography filter. At the bottom rows we match the same image to a false database image. As can be seen at the bottom left, a lot of false putative matches would arise without geometric verification, in extreme cases their count can be similar to or higher than for the correct image pair. At the bottom right all the false matches are removed, only features from the (correctly) matching frame survive and the difference in matching with the correct pair is drastically increased.*

***Figure 5.9**: Examples of query images, from left to right: (a) with compositions of text and image, (b) taken from varying viewpoints, at different camera zoom levels or may contain clutter, (c) example which selects a specific region of a slide, or (d) contains large amounts of text.*

the older 6230 model results in lower recognition rates. The results for the 640x480 database confirm the results of the 800x600 case, but yield overall lower recognition scores. This is due to the fact that at lower resolution fewer features are extracted.

	Prec. w geometry		Prec. w/o geometry	
	800x600	640x480	800x600	640x480
Nokia N70	100%	98,3%	98,3%	96,7%
Nokia 6230	97,7%	93,2%	91%	86,3%

***Table 5.2**: Summary of recognition rates for slide database.*

As mentioned above, recognition with local features also allows for the "highlighting" of parts of slides. This is especially interesting, when combined with a video stream from the phone. The movement of the phone can be tracked and an overlay can be shown on the tracked slides. An example is shown in Figure 5.10. The 320×240 feed has been matched to the slide collection, and the track has been highlighted on the identified slide. Using a camera-phone, it is possible to virtually "draw" on slides. For a real world system, some stabilization of the tracks would be required, to cope with the shaky lines created from the unstable hand-camera movement.

5.2.4 Hyperlinked Buildings: A Cityguide on a Mobile Phone

The second application for the Internet of Things we present in this chapter deals with a very different kind of objects. We "hyperlink" buildings (tourist sights *etc.*) to digital content. Thus this application forms an interface to a database like the one presented in Chapter 4. In this chapter we are particularly interested in the retrieval performance depending on the camera quality and the inclusion of multi-modal context such as geographic location information in the retrieval process.

***Figure 5.10**: Virtual highlighting of slides.*

Visual Data and Geographic Location

From the user perspective, the interaction process remains the same as in the meeting room scenario: by the click of a button on the mobile phone, a picture is taken and transmitted to the server. However, unlike in the meeting room application, the guide client-side application adds location information to the request, making the search multimodal. The geographic information consists of the current position read from an integrated or external (bluetooth) GPS device and/or the current cell-tower id CGI (Cell Global Identity).

While GPS returns longitude and latitude information, which makes localization simple, CGI needs some more explanation. The localization reflects the GSM mobile phone network and is based on the cellular structure of this network. Since the phone is connected to one or several antennae, it is possible to determine in which cell the subscriber currently is. Localization can now either build on exact positioning based on triangulation between several antennae, on rough positioning given the location of the corresponding cell-tower, or on prior observations of the same cell id at a given location [Spirito *et al.*, 2001]. The first two options are usually only possible

Element	Name	Example
MCC	Mobile Country Code	228 (Switzerland)
MNC	Mobile Network Code	1 (Swisscom), 2 (Sunrise), 3 (Orange)
LAC	Location Area Code	20000
CI	Cell Identity	26337608

***Table 5.3**: Cell Global Identity*

for network operators, since the access to the required data and functionality is not publicly available. However, some phones allow to retrieve information about the current cell, which enables us to use the third option.

The precision of this kind of cell-based positioning depends on the size of the cell. Cells in cities are small and have an extension of 200 – 300 meters, in other words, the location of mobile subscribers can be determined with a high accuracy. However, in the country-side the cells have an extension of several kilometers. (The largest cell in Switzerland has a radius of 35km.) The shape of a cell is a much more complex structure than what is generally assumed. It can be composed of dozens of geographic polygons and each polygon in turn can be composed of thousands of coordinates. Each such cell is identified by a CGI. It is created by concatenating the four elements shown in Table 5.3. Note that the LAC and the CI are operator specific, *i.e.* the same geographic location has different LACs depending on the MNC. The CGI can be obtained using the APIs of some mobile phone platforms, *e.g.* with Symbian but not with J2ME (Java platform Micro Edition). When creating a reference database, we can note the CGIs at the location of the object or the location, where the picture was taken. When comparing a query image to the database, the CGI associated with the incoming image is used to restrict the search to objects with the same CGI.

Combining a picture and location data (either GPS or CGI) forms a perfect query to search for information on static, physical objects. As mentioned before, location information alone would in general not be sufficient to access the relevant information: the object of interest could be several hundred meters away (e.g. a church tower), or there could be a lot of objects of interest in the same area (e.g. the St. Mark's square in Venice is surrounded by a large number of objects of interest). Furthermore, in urban areas with tall buildings and narrow roads, GPS data is often imprecise. On the other hand, relying on the picture only would not be feasible, either: the size of the database would make real-time queries and precise results very difficult to achieve.

After the query has been processed, the user receives the requested information directly on the screen of her mobile phone. In our demo application we open a web browser with the Wikipedia page corresponding to the object. This is illustrated in Figure 5.11.

***Figure 5.11**: Client software for the cityguide application: the user snaps a picture, waits a few seconds, and is redirected to the corresponding Wikipedia page.*

System Design

The cityguide system consists of a server side recognition system and a client-side software on the mobile phone.

The server-side elements consist of a relational database for storage of image metadata (GPS locations, cell information *etc.*) and information about the stored sights. We used mySQL for this purpose. The image recognition is implemented as a server in C++ which can be accessed via HTTP.

Queries from the client software are transmitted to the server as HTTP POST requests. A middleware written in PHP and Ruby restricts the search by location if needed and passes this pre-processed query to the recognition server. The associated content for the best match is sent back to the client and is displayed in an automatically opened browser, as shown in Figure 5.11.

Client software on the mobile phone was implemented both in Symbian C++ and Java[5]. Note that the feature extraction of the query happens on the server side, *i.e.* the full query image is transmitted to the server. Alternatively, our system can also be accessed using the Multimedia Message Service (MMS). A picture is transmitted

[5]Unfortunately, only the Symbian version allows access to the celltower ids.

to the server by sending it as an MMS message to an e-mail address. The response (Wikipedia URL) is returned as an SMS message.

Object Recognition Method

The data from the client-side application is transmitted to the recognition server, where a visual search restricted by the transmitted location is initiated. If GPS data is used, all database objects in a preset radius are searched (different radii are evaluated in the experimental section). If only cell-tower information is used, the search is restricted to the objects annotated with the same CGI string.

The object recognition approach is very similar to the method discussed for the meeting room slides. That is, putative matches between pairs of query and database images are found by nearest neighbor search for their SURF [Bay *et al.*, 2006b] descriptors. These putative matches are validated with a geometry filter.

Experiments

To evaluate the proposed method, we collected a database of 147 photos covering 9 touristic sights and their locations. The 147 images cover the 9 objects from multiple sides, at least 3 per object. The database images were taken with a regular point-and-shoot camera. To determine their GPS location and CGIs we developed a tracker application in Symbian C++ which runs on a mobile phone and stores the current GPS data (as obtained from an external bluetooth GPS device) and CGI cell information at regular time intervals. This log is synchronized by timestamps with the database photos.

We collected another 126 test (query) images, taken with different mobile phones (Nokia N70 and Nokia 6280, both with 2 Megapixel camera) on different days and times of day, by different users and from random viewpoints. Of the 126 query images 91 contain objects in the database and 35 contain images of other buildings or background (also annotated with GPS and CGI). This is an important prerequisite to test the system with negative queries, an experiment which has been neglected in several other works. Compared to the MPG-20 database[6] we have fewer objects but each of them covered from multiple sides (in total about 30 unique representations), more challenging viewpoints for each side (distance up to 500 meters), full annotation with both GPS data and celltower ids, and more than 4 times as many query images. The database with all annotations (GPS, cellids, objects Wikipedia pages etc.) is available for download [7]. Both database and query images were re-scaled to 500×375

[6] http://dib.joanneum.at/cape/MPG-20/
[7] http://www.vision.ee.ethz.ch/datasets/

***Figure 5.12**: Result images for the city-guide application, see text for details.*

pixels. (Sample images from the database are visible in Figure 5.12 and are discussed a few paragraphs below).

Note that the CGI (Cell Global Identity) depends on the network operator, since each operator defines its own set of cell ids. If the operator does not release the locations of the cells (which is common practice in many countries for privacy reasons), we have to find a mapping between the cell ids of different operators. We achieved such an experimental mapping by using our tracker application: tracks obtained with SIM cards of different mobile network operators were synchronized by their GPS locations: if GPS points were closer than 50m, a correspondence between the respective cell ids was established. This mapping is far from complete, but it simulates an approach which is currently followed by several initiatives on the Web.

	Prec. w. geometry	Prec. w/o geometry	Time
Full database linear	88.0%	67.4%	5.43s
GPS 300m radius	89.6%	76.1%	3.15s
Cell id	74.6%	73.9%	2.78s

***Table 5.4**: Summary of recognition rates for cityguide.*

We present experiments for three scenarios: linear search over the whole database without location restriction, restriction by GPS with different search radii, and restriction by cellid. For all cases we compare the trade-off between search time and recognition rate. A pair of images was considered matched, if at least 20 features matched (with and without geometry filter). From the images which fulfilled this criterion the one with the most matches was returned as a response. Table 5.4 summarizes the results. For the baseline, linear search over the entire database without geometry filter we achieve 67.4% recognition rate. This value is outperformed by over 20% with the introduction of the geometry filter, resulting in 88% recognition rate. This is due to the removal of false positive matches.

Restricting search by GPS location with a radius of 300 meters is about 40% faster while increasing precision slightly for the case with geometry filter and more substantially for the case without filter. Restriction by cell-tower CGI is slightly faster but significantly worse in precision. This seems mostly due to the fact that our CGI correspondences for different operators might be incomplete. For a real world application, where an operator would hopefully contribute the cell id information or a search radius bound by GPS coordinates we would thus expect better results.

Overall the best results are achieved with GPS and a rather large radius of several hundred meters. In Figure 5.13 we plot the precision versus time for different radii. At 100 meters we retrieve most of the of the objects correctly, but only between 300 and 500 meters we achieve the same recognition rates as for linear search, however at significantly higher speed. In fact, this speed-up over linear search will obviously be even larger, the more items are in the database. The recognition times can be further sped up with a suitable indexing structure such as the ones discussed in Chapter 6.

Visual results are shown in Figure 5.12. Section (a) shows query images in the left column and best matching database images for each query in the right column. Note the distance of the query image to the database image in the first row and the zoom and low contrast of the query in the second row. Section (b) contains a query image at the top and the best database match at the bottom. Besides the viewpoint change and occlusion through the lamp and railing, note that query and database image have very different weather and lighting conditions since they were taken several weeks apart. Section (c) shows another query database pair, this time for a facade with strong cropping and change of angle. The last image in section (d)

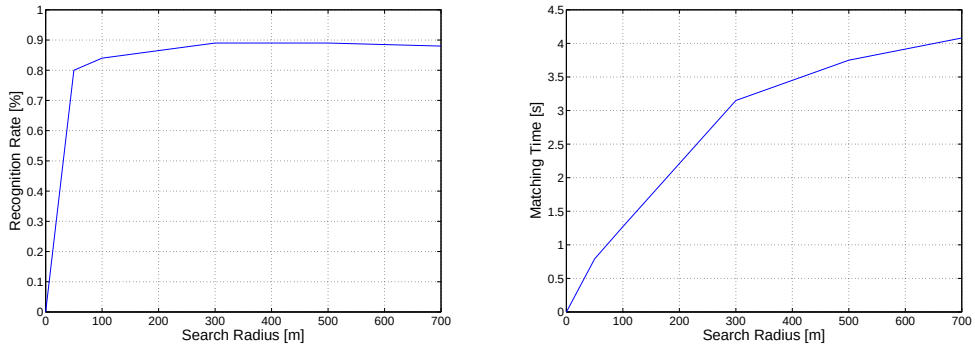


***Figure 5.13**: Recognition rate (left) and matching time (right) depending on radius around query location.*

contains a typical "negative" query image, which should not return any matching object.

The results show the beneficial effects of the geometry filter. Overall recognition rates could be improved with better coverage of database items with additional images, for instance based on the mining approach discussed in Chapter 4. Restricting search to a geographic radius of a few hundred meters increases speed significantly even in our test database and will be essential for large-scale real world applications. At the same time, the results show that relying only on GPS information (objects up to several dozen meters away) would not be suitable for a real-world guiding application. Being able to "select" from many possible objects in the the user's vicinity (including far away objects) by simply pointing the mobile phone camera to the desired target brings significant usability benefits to the users.

5.3 Object Recognition for Web Applications

As mentioned earlier in this work, the combination of the availability of cheap digital recording devices and the change of the Internet towards a more interactive, multimedial platform ("Web 2.0") opens new possibilities for object recognition applications on the Web.

Two sample applications we implemented shall serve as an example for what kind of applications are to be expected in the coming years. Our applications are an interface for auto-annotation on community photo collection, and a desktop application for 3D reconstruction of photos. Both applications build on the mining methods introduced in Chapter 4.

5.3.1 Auto Annotation for Community Photo Collections

The goal of this application is to provide a simple, web-based auto-annotation interface for photos on Flickr. The application implements the process proposed in Chapter 4.6 and build directly on the mined object clusters. The user initiates an annotation by dragging one of his photosets from Flickr to a map, as shown in Figure 5.14. The application identifies the country the set was dragged onto and tries to find annotations for the photos in the set, by matching it with the mined object clusters from that country. Currently the response time is not real-time and the annotation is not done on the bounding box level yet, but these features can easily be added.

The application is implemented using the Flickr API. Users can log-in with their Flickr username, their photosets show up, and they can proceed with the annotation process as just described.

***Figure 5.14**: Interface for auto-annotation of Flickr photos. Users start an annotation task by simply dragging a set of photos to a country as shown in the example.*

5.3.2 Browsing Photos in 3D

The ability to share images on the Web leads to collections with significant amounts of photos of the same object. We exploited this fact already in Chapter 4 by clustering photos which belong to the same object. The photos in the resulting clusters show an object from varying viewpoints, which allows reconstruction of the 3D scene

around the object. In theory, a complete and precise 3D reconstruction from pictures taken with a large variety of cameras is possible. In practice, however, this is quite challenging. An easier task consists of estimating the approximate camera positions in 3D-space. This allows browsing photos in 3D, as proposed in [Snavely *et al.*, 2006]. The focus of that particular work is on the 3D reconstruction process The creation of the photo-clusters the reconstruction is based on is not discussed. The combination of our mining work with 3D browsing as in [Snavely *et al.*, 2006] thus offers the potential for exciting user interfaces to data in community photo collections.

To investigate this potential in some more detail, we created a simplified implementation of the system described in [Snavely *et al.*, 2006]. Our system consists of a "classic" Structure-from-Motion (SfM) pipeline [Hartley and Zisserman, 2004]. The main challenges in our setting are estimating the correct intrinsic camera parameters from EXIF data provided with the photos, and the selection of a "good" starting pair of images for the SfM process. Sensor data required to calculate the internals for each camera is read automatically from EXIF files, from a database downloaded from the Web, or by crawling camera datasheets from the Web. The starting pair is selected by creating "tracks" of features through matched images, as proposed in [Snavely *et al.*, 2006]. (Our implementation uses connected component analysis on a graph of matched features to identify good feature tracks). Point correspondences for 3D-reconstruction were calculated from Harris corners [Harris and Stephens, 1988], due to their more accurate localization compared to the Hessian used in the SURF interest point detector. The implementation of the required framework was carried out by Fabio Magagna during his MSc thesis, which summarizes the details of the approach in [Magagna, 2008].

Figure 5.15 shows sample results. The examples demonstrate that it is possible to calculate correct 3D representations for browsing from the data acquired with the method from Chapter 4 – without additional supervision. However, full 3D reconstruction of objects, especially for smaller clusters and for more complex items than the ones shown in Figure 5.15 would require substantial additional refinements at the 3D reconstruction pipeline.

While writing this thesis, a work which shows very similar results appeared in [Li *et al.*, 2008b].

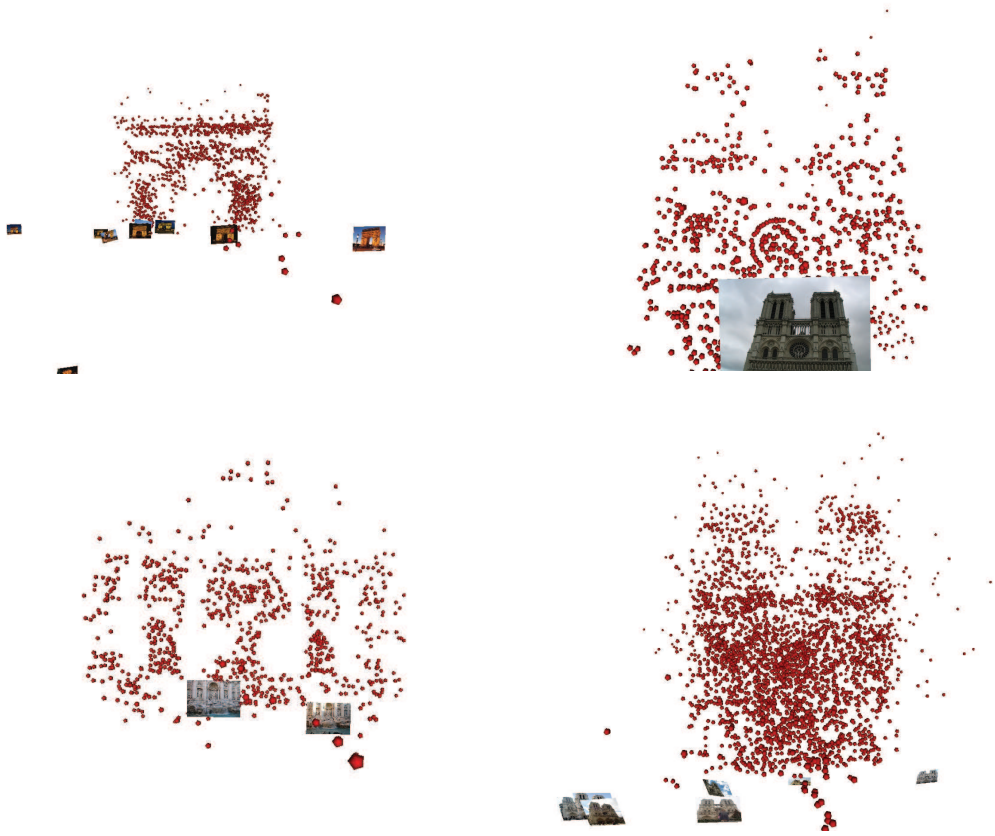


Figure 5.15: *Examples of 3D reconstruction from community photo collection data. The photos are positioned at the estimated camera locations, the object is represented by a point-cloud calculated from point correspondences. In clockwise direction: Arc du Triomphe, Notre Dame de Paris (close-up), Notre Dame de Paris, Trevi Fountain Rome. (Only a small selection of cameras is shown)*

***Figure 5.16**: Examples of text in natural scenes*

5.4 Detecting and Reading Text in Images

In the preceding sections we described methods and showed implementations of systems that mine or retrieve certain types objects, *e.g.* landmark buildings from digital repositories such as online community photo collections. Many of those images also contain another very specific type of "object", namely text. Text is omnipresent in our surroundings – it appears on street signs, store signs, product packagings, *etc.* A good retrieval system for visual data would make use of this information by extracting and indexing it. This would allow multimodal retrieval based on text keywords, or potentially other applications, such as geotagging an image based on a street name which was read from a street sign present in the image.

However, unlike text in scanned documents, the text in those natural scenes can appear anywhere in the image, in any font, at any scale, with (perspective) distortions, variable lighting, and large amounts of clutter. Figure 5.16 shows a few examples of text in images from `flickr.com`. Nevertheless, text has very characteristic visual properties. Thus, one might try to learn these properties and use them to locate text in images such as the ones from Figure 5.16. In a second step, OCR (Optical Character Recognition) could be applied on the detected regions, and make the extracted text available to a retrieval system.

We propose an approach for text localization based on the Viola-Jones face detector [Viola and Jones, 2001b]. This is motivated by the observation that text, just like faces, seems to be composed of quite simple and characteristic patterns. The hope is thus, that adapted features might be learned well with an approach similar to face detection.

This chapter summarizes a joint work with my former Masters student Martin Renold. His report [Renold, 2008] contains further implementation details and additional interesting evaluations.

5.4.1 Text Detection Approach

Our text detection approach is an adaption of the Viola-Jones method for face detection [Viola and Jones, 2001b]. Their system consists of a trained classifier for faces, which is evaluated on rectangular windows at all locations and scales to locate faces in previously unseen images.

To make this extraordinary large detection problem solvable in reasonable time, their well-known method relies on a few key ideas, which are summarized in the following:

- Image features are based on rectangular blocks and can be computed using so-called *integral images* at any scale in constant time.
- A simple threshold is used to treat a feature as a *weak classifier*.
- *Adaboost* is used to automatically select a subset of all weak classifiers and to combine them into a strong classifier.
- A *cascade* of increasingly complex strong classifiers allows to reject easy background early on without sacrificing much computation.

We discuss our slightly modified version of this pipeline for text detection in the following sections.

5.4.2 Features

Text detection differs from face detection in several aspects, which leads to the following requirements for our features:

- the features should be invariant to color inversion, since both black on white and white on black text should be detected.

***Figure 5.17**: Block based features are parameterized by their location and size. All possible rectangles within this 10x10 raster are considered during training.*

***Figure 5.18**: The intensity based features used. The absolute intensity difference between the black and the white region is calculated. Left: comparing block to window intensity; right: Haar-like edge features*

- while frontal face detection may use well-located features of the face (like the eye region) the position and shape of letters are not fixed. Therefore statistical properties (*e.g.* the texture) are more important.

We thus propose to use four types of features based on intensity, variance, edges, and a scanline property. These features are calculated as follows: Similar to [Viola and Jones, 2001b] most of our features are based on calculating measures on the pixels within a sub-rectangle R of the window W, which is to be classified as text or non-text. The geometry of this sub-rectangle is parameterized as in Figure 5.17. Figure 5.18 shows three intensity-based features calculated from such sub-rectangles. These features are similar to the ones used in [Viola and Jones, 2001b]. The main purpose of the first feature (full block), is to check whether the regions above and below the text line have a color different from the text. The horizontal and vertical Haar-like features (2nd and 3rd in Figure 5.18) are also used in [Viola and Jones, 2001b]. However, we use absolute values in order to detect black and white text equally well. Both feature types are contrast normalized with respect to the whole window.

Another simple feature is the variance of the gray level values of pixels inside a region. It is motivated by the observation, that regions with very low intensity variance are quite common (sky, uniform surfaces) and rarely contain text. Furthermore, the

feature can be calculated efficiently from the squared integral image. We used two variants of this feature type. The first feature is simply the variance inside the whole detection window. Conveniently this value needs to be calculated anyway to contrast-normalize intensity based features. The second feature calculates the ratio between the variance inside a subrectangle and the variance of the whole window. This has the effect of contrast normalization. One property of this feature is, that it checks, whether a subregion contains only bright or only dark pixels instead of text.

A third type of features is based on edges. The idea is that text has a certain minimum and maximum number of edges. Those statistics are different for horizontal and vertical edges, and also depend on the position within the detection window (*e.g.* there is usually a blank stripe above and below the text). The challenge is to somehow "count" the number of edges with just a few lookups in an integral image. To that end, we first perform edge detection on the original grayscale image $I(x, y)$ using the Sobel operator. This results in the gradient images $G_x(x, y)$ and $G_y(x, y)$:

$$G_x(x, y) = I(x, y) * S_x(x, y) \tag{5.1}$$
$$G_y(x, y) = I(x, y) * S_y(x, y) \tag{5.2}$$

with

$$S_x = \frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ and } S_y = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{5.3}$$

For the next processing steps, only the absolute values $|G_x(x, y)|$ and $|G_y(x, y)|$ are

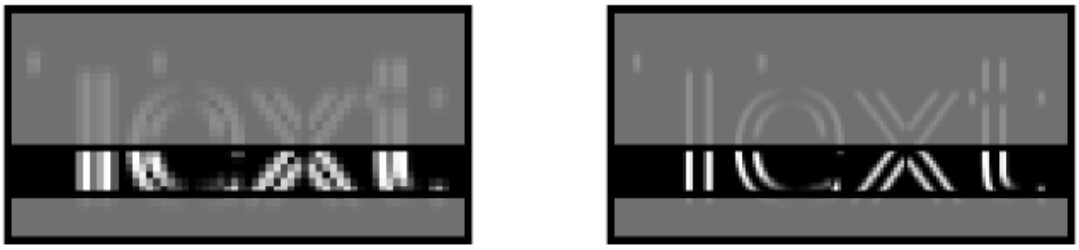


Figure 5.19*: The values of $|G_x(x, y)|$ of the same text at different resolutions. We are interested in counting the number of vertical edges inside the horizontal stripe region R, independent of the resolution.*

considered. Let us further assume that the original image is a clean black-and-white binary image. Figure 5.19 shows an example of the $|G_x(x, y)|$ values in a rectangular block R. The sum along a single horizontal pixel row

$$g(R, y) = \sum_{x \in R_x} |G_x(x, y)| \tag{5.4}$$

is now a good approximating of the number of vertical edges crossing this row. When combining all h pixel rows inside the $w \times h$ rectangle R to get a more robust feature, the values of $g(y)$ have to be averaged to still get an equivalent to the number of edges:

$$e_x(R) = \frac{1}{h} \sum_{(x,y)\in R} |G_x(x,y)|$$
$$e_y(R) = \frac{1}{w} \sum_{(x,y)\in R} |G_y(x,y)|$$

Because the text will rarely be clean black on white, e_x and e_y depend on the contrast of the text and the amount of noise in the window. Thus, three normalization methods were put into the feature pool for Adaboost to choose from. For the contrast:

$$\text{window contrast: } f_1 = \frac{f_x(R)}{s(W)} \tag{5.5}$$
$$\text{block contrast: } f_2 = \frac{f_x(R)}{s(R)} \tag{5.6}$$

where $s(R)$ and $s(W)$ are the standard deviation of the block R and window W, respectively. To tackle the noise problem, a third normalization comparing to the number of edges within the whole detection window was used:

$$f_3 = \frac{f_x(R)}{f_x(W)}. \tag{5.7}$$

Combining the three normalization methods f_1, f_2 and f_3 with e_x, e_y and $e_x + e_y$ results in nine different feature types. As a tenth edge based feature, we used the fraction of the horizontal edges:

$$f_r = \frac{f_x(R)}{f_x(R) + f_y(R)}. \tag{5.8}$$

Finally, a last feature type – scanline based features – was discovered rather coincidentally. Starting from a border pixel of the detection window, all pixels along a given scanline are walked through at the full image resolution (Figure 5.20). The idea is to find the minimum or maximum segment length in the binarized image. Binarization is done with a hysteresis to reduce noise effects near the transitions.

The high and low threshold values are centered around the window mean intensity, with a distance chosen by Adaboost (between 0.5 and and 2 times $s(W)$).

We will show in the experimental section that adding each of the introduced feature types improves recognition rates.

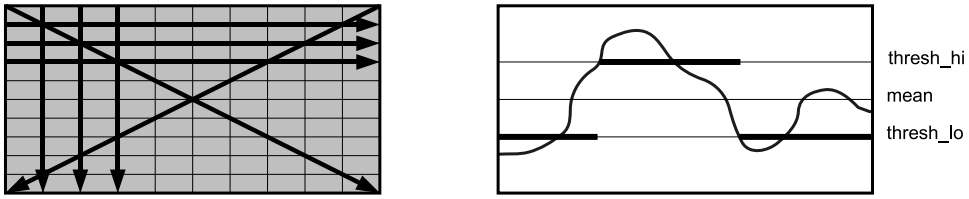


Figure 5.20: *10 horizontal, 10 vertical and 2 diagonal scanlines were in the feature pool. The intensity value is tracked and binarized (with hysteresis) along the selected scanline. The result is the minimum or maximum distance between two transitions.*

5.4.3 Classifier Training

Training the feature cascade using boosting requires labeled training windows as input. On training images, readable text lines were thus manually labeled with rectangles, including some space above and below the letters. These labels were split into detection windows with width-to-height ratio 2:1 and then used as positive samples for Adaboost. An example annotation is shown in Figure 5.21.

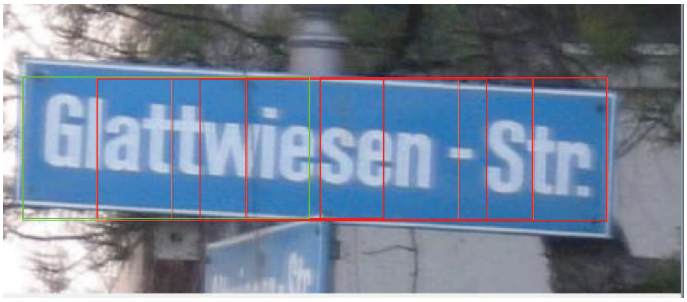


Figure 5.21: *Annotation Sample: overlapping* 2×1 *windows used for annotation, one example window is marked in green.*

Bootstrappingg was done by randomly sampling background windows until their number was equal to the number of the foreground windows. The last stage of the cascade was allowed to train with only half as many background samples. The training stops when there is not enough background left in the training set.

Classifier training follows mostly the approach proposed by [Viola and Jones, 2001b]. The only significant difference in our implementation is the use of Discrete Adaboost with a modification for asymmetric learning, which updates the weights for each boosting round in an asymmetric way. It is based on an approach proposed in [Viola and Jones, 2001a] for Asymmetric Adaboost, but applied to Discrete Adaboost: before each boosting round the weights of the positive samples are multiplied by a

factor C and the weights of the negative samples are divided by C. The factor C is defined as

$$C = exp(\frac{k}{T}) \tag{5.9}$$

where T is the total number of boosting rounds and k a constant chosen by the user. Note that $k = 0$ stands for the symmetric case. Good choices for k turned out to be $1 < k < 4$.

One problem that arises here is that the number of boosting rounds T has to be known before the boosting starts. Because the performance on the validation set is used as a stopping criterion, T depends on how well the selected features work. But this depends again on the choice of C. To resolve this, each stage is trained twice: once with T set to the number of boosting rounds of the previous stage, and once again with T set to the result of the first training round. This could be repeated several times until T converges, but one iteration seems to be enough for practical purposes.

5.4.4 Detection and Reading

Detecting text in a novel image is carried out by executing three steps:

1. Detect windows containing text-fragments using the trained classifier
2. Combine detected windows into lines of text
3. Read the text lines using OCR

Each of these steps is described in the following.

Detecting text fragments

For a test image all windows of all scales are classified in either text or non-text using a classifier cascade trained as described in the previous section. A minimum text window size of 40×20 pixels was used. The maximum size of the scanning window is limited only by the image dimensions. Scanning windows are set at steps of 0.4 (horizontal) and 0.2 (vertical) of the window size at the current scale.

Combining text fragments

The raw detection windows are clustered using a greedy window merging approach. Isolated detections are discarded, assuming that they are false positives. A detection rectangle R_1 is merged to the cluster of the rectangle R_2 if their intersection-over-union measure is higher than 0.4:

$$\frac{|R_1 \cap R_2|}{|R_1 \cup R_2|} > 0.4 \tag{5.10}$$

The merging procedure between windows and clusters is the same as the one used in single-link hierarchical agglomerative clustering. The prior knowledge that text detections are more likely to cluster horizontally than vertically is used by enlarging all raw detection windows horizontally by 1/3 of their original width before clustering. This turned out to have the additional benefit of including the first or last letter of a text-line, which were often missed before.

Clusters consisting of less than three detection windows are discarded. To remove outliers, for each of the remaining clusters, the text height is estimated by the log-average within the cluster:

$$\hat{h} = exp(\frac{1}{n}\sum_{i=1}^{n} log(h_i)) \tag{5.11}$$

The final text bounding box created from each cluster is the union of all detection windows, excluding windows that are more than one scale step above the estimated height $\hat{h}$. This modification was added due to the observation, that including all detection windows often lead to an over-estimated bounding box size.

Reading Text using OCR

The final step for making the detected text accessible to a retrieval system consists of decoding it using Optical Character Recognition (OCR). For that purpose we rely on existing OCR engines. Results obtained using several commercial and open-source engines are presented in Section 5.4.5.

Due to the challenging nature of the text we extracted from images of natural scenes, several pre-processing steps are necessary, before feeding a text window into an OCR engine. First, the detected regions are cut out and their histogram is normalized. In order to keep processing times low, the resulting cropped image is scaled down to a maximum height of 80 pixels.

Most OCR programs accept greyscale images as input, however earlier works such as [Chen and Yuille, 2004] reported to get better results with prior binarization of

text areas. In general, binarization is a well studied topic in Computer Vision, a recent study for the specific task of text binarization under challenging conditions is for example [Lu and Tan, 2007].

Note that most binarization methods assume a black on white text and will give very poor results for white on black. Because of this, white on black images must be detected and inverted before binarization. Since there are usually more background pixels than text pixels, this can be done simply by counting the pixels below the mean intensity. The image is inverted if more than 50% of the pixels are below the mean intensity.

We considered three different thresholding approaches for binarization: Otsu's global thresholding, Niblack's adaptive thresholding and Sauvolas algorithm which is a variant of Niblack (details can be found in [Lu and Tan, 2007]). Figure 5.22 shows an example with the three thresholding methods applied. It is evident that in contrast to scanned text, for natural scenes, global thresholding is not an option, since it can't handle effects such as gradients.

***Figure 5.22**: Original image, global threshold (Otsu), local threshold (Sauvola).*

5.4.5 Experiments and Results

We present results of our method on a series of datasets, with a focus on detection precision, but also reporting results on the whole recognition pipeline including OCR performance. Three different datasets were used as training and testsets:

FlickrText. We collected a text dataset consisting of pictures of street signs and advertisement panels in the region of Zürich. This set also contains images of book pages, newspapers, and a few URLs and numbers displayed on LCD screens. Additionally about a quarter of the images were downloaded from Flickr[8]. Only text in roman letters was collected and annotated. In total, we labeled 599 rectangles in 209 images split into 3423 detection windows.

The background regions were also labeled manually in order to have text, non-text and unlabeled data. Unlabeled areas were necessary for special cases, in particular small, unreadable, rotated and heavily distorted text, as well as artistic fonts and graffiti. It turned out that the training process often stopped because it could

[8]www.flickr.com

not find enough false positives to train on. Thus, additional city scene images were collected from Flickr, and examples which contained many false positives were labeled as additional background training data. Alltogether, background regions from 632 images were thus used. Images from the Flickr set were then split randomly into three subsets of equal size for training, validation and testing.

CamPhoneText. For this set, we took pictures of signs, newspapers, screens, *etc*. using low-quality mobile phone cameras. We annotated the readable text in 88 challenging low-quality images (640×480 with blur and noise). This set is intended as a particularly challenging test set.

ICDAR. The ICDAR set is a benchmark set[9] used in the 2003 and 2005 Text Locating Competitions [Lucas, 2005] of the International Conference on Document Analysis and Recognition (ICDAR). The set consist of 258 training images and 251 test images.

We evaluate the performance of our approach by measuring the quality of the text detection and text reading separately.

Text detection

Text localization is measured by evaluating the correct detection of the *individual* 2:1 letter annotation windows. We evaluate the results using ROC curves, plotting correct detections vs. false positives. The *false positive rate* is simply measured by running the detector on the labeled background regions in the testing set. The positive responses are counted and divided by the total number of classified windows. This way, detections that overlap with a text region are never counted as false positives.

For the true positive rate, we used two measures. The *strict hit rate* requires that each window is detected at its exact position. The *permissive hit rate* (or just *hit rate*) counts a true positive for the ground-truth window A, if there is any text window B detected with an intersection-over-union value higher than 0.4:

$$\frac{|A \cap B|}{|A \cup B|} > 0.4 \tag{5.12}$$

This measure is more realistic, since the detection of individual letters does not need to be perfect, as clustering merges the letters later. Note that we use a value of 0.4 for the threshold, which is lower than in other object recognition tasks, where it is

[9] http://algoval.essex.ac.uk/icdar/Datasets.html

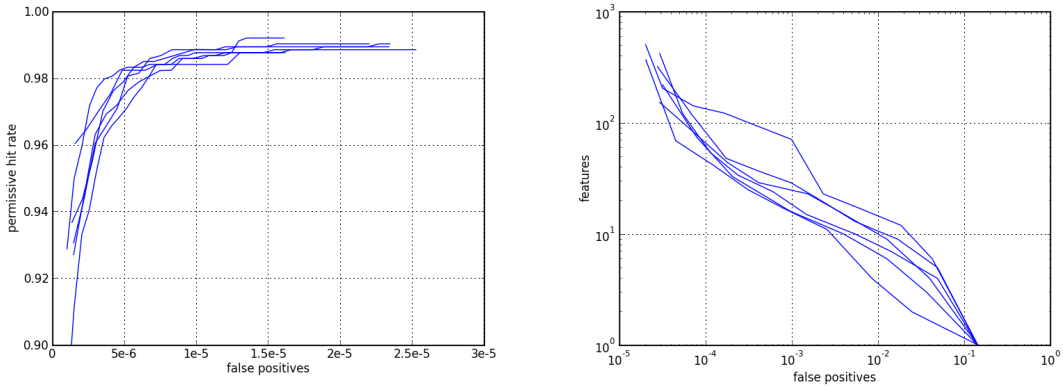


***Figure 5.23**: Detection results on Flickr set. Left: ROC curves for six training runs. Right: complexity of the cascade for the same training runs (expressed in the number of features used by the classifier).*

typically 0.5. The reason is that the small letters contain more uncertainty already in their annotation.

Overall detection results are shown in Figure 5.23. The ROC curves are created by varying the Adaboost threshold of the last stage. The different outcomes for the same training run are due to the random sampling of background during the training process. The overall recognition rate is with 98% extremely high. Compared with the state-of-the-art reported in [Lucas, 2005; Chen and Yuille, 2004] it seems we reach the same level of precision, however, direct comparison is difficult, since different evaluation measures on different datasets are used. (The evaluation is on a detected word level - our algorithm detects individual text windows, which are then merged to lines of text, but no discrimination of individual words is attempted at this stage). It would be interesting to test our system in a new benchmark like [Lucas, 2005], by extending our algorithm to extract individual words.

The total runtime to produce the raw detection results on a typical 1600×1200 JPEG image is about $0.5s$ (Pentium 4, 2.40GHz). The actual detection part (after the integral images are calculated) takes $0.18s$. Figure 5.24 shows raw text detection windows in difficult natural scenes.

Figure 5.25 shows how the various feature types are selected and combined. Each of them leads to a substantial precision improvement while at the same time reducing classifier complexity. It is worth noting that in the first four stages Adaboost selects only edge based features. In the later stages all feature types are selected, with a slight preference for edge based features.

Figures 5.26 and 5.27 show example results on the ICDAR trial test set. The results show the detected text after clustering the individual detection windows. Figure 5.27 shows some typical false positive windows. It is also typical, that multiple fasle

Figure 5.24: *Raw detection results: All results correspond to the highest ROC curve in Figure 5.23. Note the handwritten text in the topmost line, rotated text, and gradient in the 2nd and 3rd image, respectively, and the large amount of text handled in the last example. The number at the bottom is not detected due to the lack of spacing to the barcode.*

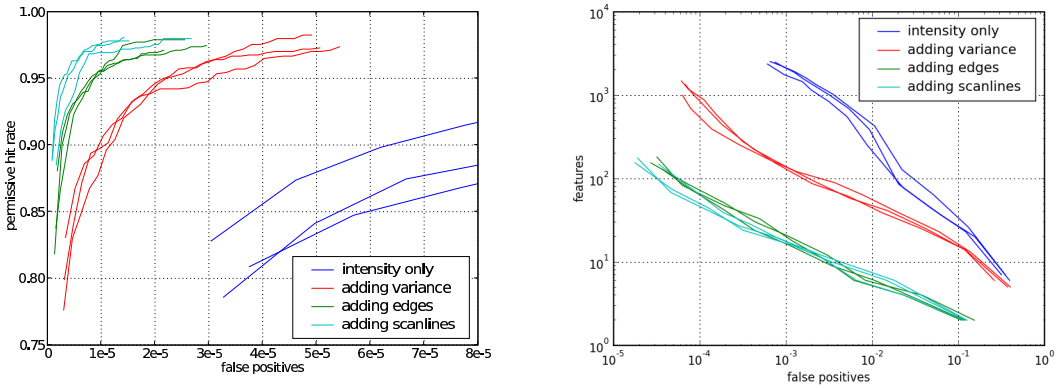


Figure 5.25: *Feature Selection and Combination by Adaboost. Effect on detection performance (left), and classifier complexity (right).*

Figure 5.26: *Two difficult text areas from the ICDAR trial test set. True positives in red and false positives in blue. Operating at a higher false positive rate* ($45 * 10^{-6}$) *allows to find more of the text.*

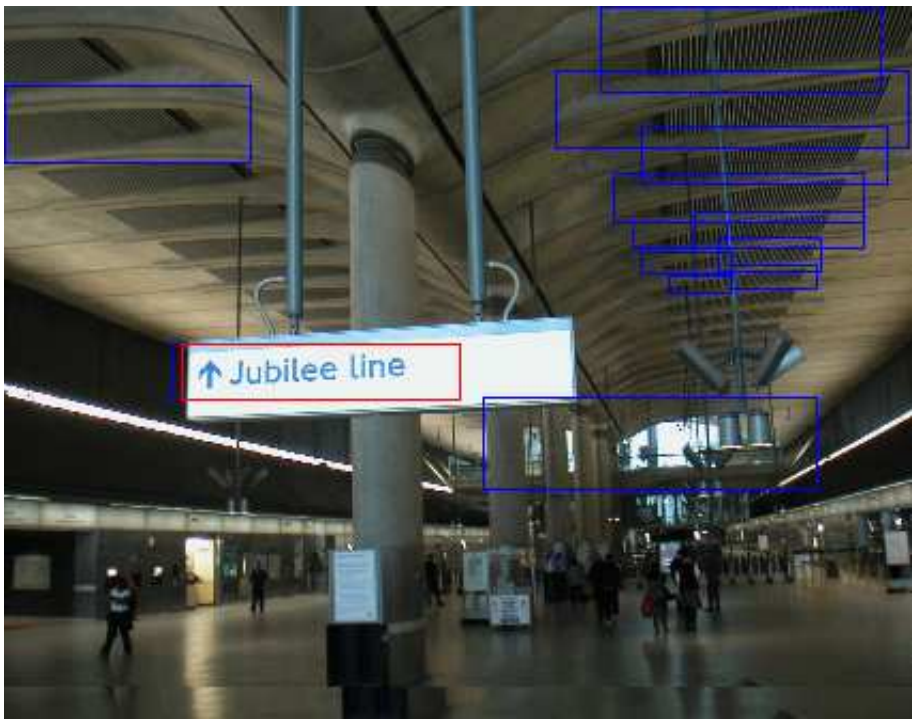


***Figure 5.27**: False positives. Some typical false positives when running at $45*10^{-6}$ false positives (blue, background). At $5*10^{-6}$ false positives (red) only the correct text is found.*

positives appear at the same location. Both figures demonstrate the tradeoff that has to be found between tolerating false positives and missing some of the more difficult text.

Figures 5.28, 5.29 and 5.30 shows additional example results for the text detection after clustering raw detections.

Reading Text

To measure the OCR performance, the ground truth and the OCR output for each image were treated as a "bag of words". This is based on the assumption that a word has been located correctly, if it was read correctly. We counted the number of true positives and false positives. The true positives where counted with a strict measure for correctly read words and a softer measure counting "almost correctly" read words. "Almost correct" is defined as words with a Levenshtein [Levenshtein, 1966] (edit) distance to their groundtruth counterpart smaller than one third of the correct word length.

All detected words, that do not refer do any word in the groundtruth we call "clutter". (Essentially, these are false positives).

Figure 5.28: *Example detections (I). True positives in red, false positives in blue. Note the variety of viewpoints and fonts, including handwritten text.*

***Figure 5.29**: Sample detections (II). Note the text appearing in different contexts, including station lists, cell-phone screens, cars, and stores.*

Evaluation was carried out mainly on the CamPhoneText set, and on the ICDAR set. We used the very challenging CamPhoneText, since it simulates a mobile application scenario, where users send in text photographed with their mobile phone, with the goal to initiate a web-search based on the extracted text.

Three different OCR engines were tested, two of them free software and one commercial: Tesseract[10], GOCR[11] and the ABBYY FineReader Engine (FRE)[12]. The results on the ICDAR set are shown in Figure 5.31 (left). FRE outperforms Tesseract in terms of quality, however Tesseract is faster. Overall, the rate of correctly read words is with 8% quite poor, which is surprising after the good localisation results observed earlier. Unfortunately, the ICDAR competition [Lucas, 2005] did not evaluate reading of text, such that no means of comparison is given. The absolute performance on the CamPhoneText dataset Figure 5.31 (right) is drastically lower

[10]http://code.google.com/p/tesseract-ocr/
[11]http://jocr.sourceforge.net/
[12]http://www.abbyy.com/

Figure 5.30: *Sample detections (III)*

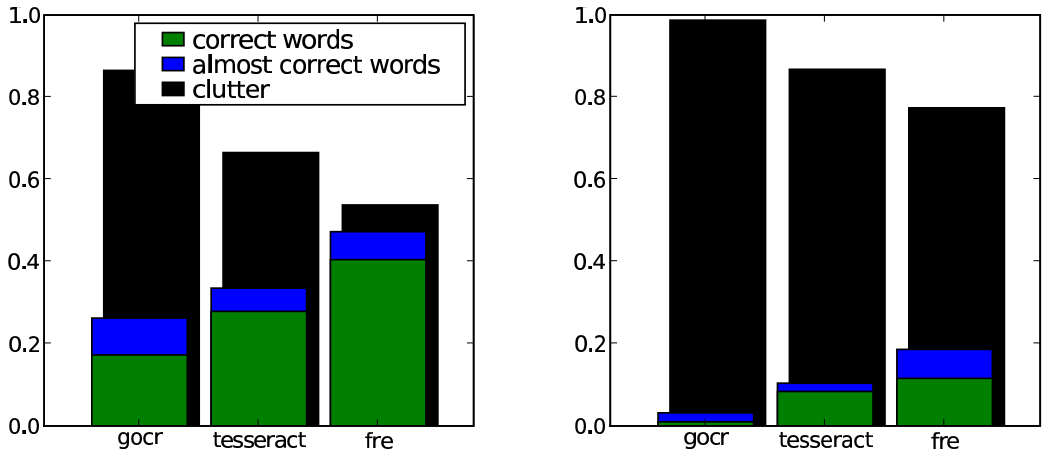


Figure 5.31: *Comparing the different OCR engines after Niblack adaptive thresholding, on the ICDAR datasets (left) and on the CamPhoneText dataset (right). This is the result of the complete system, meaning that a missed word can be either unreadable or not located.*

due to the low resolution and heavy blur of the text in most images. While some text was not found by the detector, much of the correctly located text could not be read due to the blur.

In contrast to the PhoneCamText set, the ICDAR set contains high quality and high resolution images. Missed detections were rather due to special fonts, cluttered backgrounds and single letters or digits. Our detector requires a minimum of about three letters, and the FlickrText training set did not include many images with special fonts. An experimental training of the detector with a more challenging dataset resulted in a more complex classifier, without a significant improvement in precision.

As a verification, we also feed the whole image (without a localization step) to the OCR engine. The output were only false positives in most cases.

The effect of binarization can best be seen with Tesseract in Figure 5.33. Niblacks method turned out to work best, possibly because the advantage of Sauvolas method would be mainly on empty regions, which do not appear often within the well-located text boxes. Both implementations had two additional hard (non-adaptive) thresholds for very dark and very bright regions, suppressing the most obvious noise.

***Figure 5.32**: A sample image from the low quality dataset. Using FRE, initially only the text "QZH-737695" was returned. The text "www schmdJer comm" could be read after scaling up the image.*

Unlike Tesseract, FRE performed almost equally well on the original greyscale image as with Niblack binarization (see Figure 5.33 (right)). Scaling the image up did help sometimes, for example for the image shown in Figure 5.32.

Once text in an image has been located and read this opens a wealth of possibilities for improved retrieval applications. One entertaining example is shown in Figure 5.34. The task is to guess the location a picture was taken at by the text

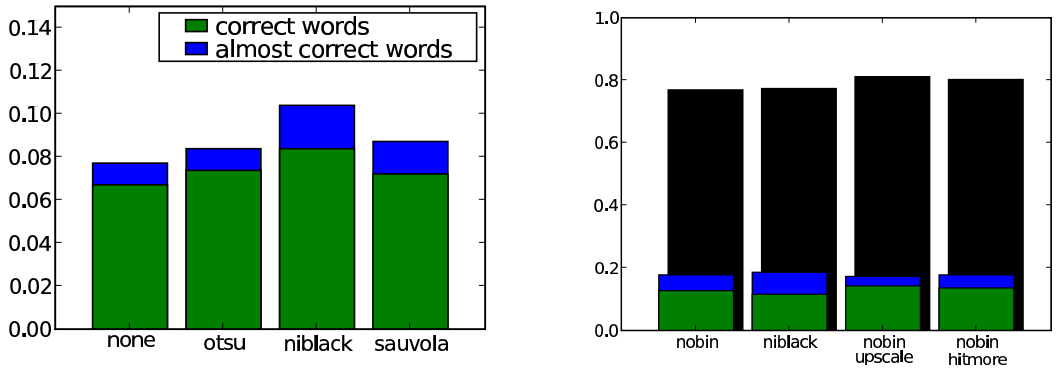


***Figure 5.33**: Left: Results with Tesseract on the PhoneCamText set using the raw greyscale regions, using global thresholding (Otsu), and using an adaptive threshold (Niblack and Sauvola). Right: Comparing different methods on the PhoneCamText set with the FRE Engine: without binarization, Niblack binarization, resized images, and different operating point of detector (higher true positive rate).*

detected in an image. In this example the location (Haapse, Estonia, where a large fraction of this thesis was written) is correctly identified.

In summary, our boosting-based approach for text detection showed very good results with EER of over 90% on images of natural scenes. The combination of the proposed features led to a strong and effective classifier, while maintaining fast execution times. It turned out, however, that reading the text from the extracted regions poses substantial challenges to current OCR engines. Especially on low quality images taken with phone cameras, the result of OCR due to blur and clutter is of very low precision. In conclusion, currently text recognition in natural scenes is only feasible for images of high quality and by employing powerful OCR techniques on the detected text windows.

***Figure 5.34**: Guessing location from text in images. (a) Where was this picture taken? Detected text is labeled with the bounding box. (b) Close-up of the text. (c) After binarization. (d) Text returned from OCR. (e) Result when sending the text from (d) to Google maps. The picture was taken by the author of this thesis in the small village Haapse, Estonia, indeed.*

5.5 Related Work

Our work on retrieval relates to a wide range of work carried out in this field. In the widest sense, it relates to the early works on image retrieval mentioned in the introduction of this chapter. Our work on retrieval for mobile devices relates to more recent work in several aspects. One aspect covers work related to our smart meeting room application, for instance the use of camera-equipped mobile phones as an interaction device for large screens. Here, Ballagas et al. have suggested a system [Ballagas *et al.*, 2005] which allows users to select objects on large displays using the mobile phone. However, their method relies on additional 2D barcodes to determine the position of the camera and is meant to use the mobile phone like a computer mouse in order to drag and drop elements on the screen. Very recently, in [Boring *et al.*, 2007] a system similar to ours has been proposed for recognizing icons on displays. While the screens are conceptually similar to the ones used in meeting rooms, we are not aware of any other work that has proposed using camera-equipped mobile phones for tagging or retrieval of slides in smart meeting rooms. The most similar works in that respect deal with slide retrieval from stationary devices. For instance [Vinciarelli and Odobez, 2006] have proposed a system, which applies optical character recognition (OCR) to slides captured from the presentation beamer. Retrieval and browsing is done with the extracted text. In contrast to our work, the method cannot deal with illustrations or pictures in the slides. SlideFinder [Niblack, 1999] is a system which extracts text and image data from the original slide data. Image retrieval is based on global color histograms and thus limited to recognize graphical elements or to some extent the global layout of the slide. Using only the stored original presentation files instead of the captured image data does not allow for the synchronization of the slides to other modalities such as recorded speech or video. Both systems are only meant for query-by-keyword retrieval and browsing from a desktop PC. While our system could also be used for off-line retrieval with query-by-example, we focus on tagging from mobile phones. This requires the identification of the correct slide reliably from varying viewpoints, which would not be possible with the cited approaches.

Another aspect is covered by work on guiding applications on mobile devices. [Bay *et al.*, 2006a] have suggested a museum guide on a tablet PC. The system showed good performance in recognizing 3D exhibition objects using scale invariant local features. However, in their system the whole database resisted on the client device, which is generally not possible for smaller devices such as mobile phones and larger databases. A similar system on a mobile phone, but with somewhat simpler object recognition is the one proposed in [Föckler *et al.*, 2005]. The suggested recognition relies on simple color histograms, which turns out not to be very robust to lighting changes in museum environments. Discriminating instances of the objects in our

applications, namely slides or outdoor images of touristic sights, is even less reliable with global color histograms.

The work most similar to our mobile city guide application is maybe [Paletta *et al.*, 2006]. Similar to the cityguide application presented in this paper, the authors also suggest a cityguide on a mobile phone using local features. However, their focus is on improving recognition capabilities using informative and compact iSift features instead of SIFT features. Our work differs significantly in several points: we use multiple view geometry to improve recognition, we rely on SURF features (which are also more compact and faster than SIFT features), and we also investigate numerically the effects of restriction by GPS or cell ids on the recognition rate and matching speed. Finally, the test databases we propose contain images taken from viewpoints with much larger variation than the databases used in [Paletta *et al.*, 2006].

Text extraction from natural scenes (Section 5.4) has also been covered by a small number of recent works. An overview of early text information extraction systems can be found in [Jung *et al.*, 2004]. More recently [Chen and Yuille, 2004] proposed an approach very similar to ours. It is worth mentioning that this system was ranked second in the ICDAR 2005 Text Locating Competition [Lucas, 2005], only being 2% lower in precision than the first-ranked method, but 40 times faster. A similar approach was also taken in [Wu, 2005], where text localization was part of a spam classification system. While their methods are also based on the concepts from [Viola and Jones, 2001b], our method differs in several ways. First, we propose a different set of features to detect text. Furthermore, we evaluated feasibility of text detection on data taken with mobile phone cameras, with the goal to extend our object recognition system with text recognition on mobile devices. Finally, preliminary results not included in this thesis indicate, that our approach can be also used to efficiently detect and decode 2D barcodes in natural scenes.

5.6 Discussion and Conclusions

We have presented retrieval applications for multimodal scenarios. The applications focus on identifying a specific object in a scene and to return related multimodal information about the identified object. The required information about objects can be collected *e.g.* with a mining process, as presented in Chapter 4.

We have put a strong focus on object recognition for mobile phones, which allows users to request information on objects by taking a picture of them. Our approach to object recognition for mobile devices relies on server-side recognition, combined with an optimized user interfaceon the client-side. On the server-side, we have implemented a recognition system and evaluated its capabilities in two challenging

scenarios: slide tagging from screens in smart meeting rooms and a cityguide on a mobile phone. For both applications the object recognition system is based on state-of-the-art local features, combined with a geometric verification of potential matches. Multimodal information such as the geographic location of the mobile user are added to the query process in order to increase precision and scalability. Evaluation carried out for both applications showed the benefits of using a geometric verification, while recognizing both planar slides and 3D buildings from challenging images taken with mobile phone cameras.

For the mobile user interface, we have investigated several options, including real-time streaming and augmentation of objects on the screen with 2D bounding boxes, and a motion detection based interface for automatic initiation of queries to the server. Especially the motion detection based approach seems interesting for further evaluation. Identifying an object from databases with millions of items on the device itself will probably be infeasible for at least a few years. Thus, server-side recognition offers significant advantages. The load on the server can be reduced by iniitating queries sporadically. On advanced mobile phones such as the iPhone client-side tracking of detected objects could now be added, which would result in an improved user experience and would allow for interesting mobile augmented reality applications.

In addition to the mobile user interfaces we also introduced two sample applications for the Web. One application exhibits a simple drag-and-drop user-interface for auto-annotation of photos in community photo collections with detected landmark buildings *etc*. The second application reconstructs a 3D point cloud and camera position estimations from clusters of photos belonging to the same object. Such an application is intended for appealing browsing of photos in community photo collections. Both applications build directly on the results from the mining method presented in Chapter 4.

Finally, we have also proposed an approach to localize text in images of natural scenes. This has applications for both mobile- and web applications, allowing to access the text in the images for keyword search, or conversely, using it to initiate text queries from images in a QBE scenario. The approach is based on the Viola-Jones approach for face detection using modified features which obtained good results on challenging benchmark datasets.

6

Scaling Retrieval

6.1 Introduction

In the preceding chapters we introduced several methods for object-level mining and retrieval of visual data. All of these methods have in common, that they rely on local image features for recognition. The recognition process always includes a matching step, where, for a query feature vector, the matching vectors from a database have to be identified. To make a mining or recognition system scalable, this matching step has to be efficient. One way to achieve scalability is to reduce the amount of data that has to be searched by including background knowledge. We did this in Chapter 4 when we carried out matching of images per geographic tile. And in Chapter 5 we reduced the search space for landmark buildings by including geographic location data with the query. However, sometimes we can't avoid searching databases with millions of items. Either because the data does not offer any possibility to restrict search with another modality (*e.g.* a database of book or CD covers), or when even with the inclusion of restrictions the number items to be searched are in the order of hundreds of thousands. Finally, for many real-world applications interactive response times and the ability to process queries from multiple users efficiently are desired. These cases require scalable methods to search large databases of local features.

Using visual vocabularies (see Chapter 2.3) has recently been shown to scale to large amounts of data, when using large vocabulary sizes [Nistér and Stewénius, 2006; Philbin *et al.*, 2007]. However, significant amount of research has been carried out over many years trying to solve the underlying general problem, namely the efficient identification of (approximate) nearest neighbors in high-dimensional spaces. Some of these methods promise advantages over the visual words approach, mostly by avoiding the time consuming clustering process, which is required to create visual vocabularies. The goals of this chapter are thus twofold: first we want to investigate the performance of alternative methods to the clustering approach, especially methods which would in theory promise better scalability than k-means clustering.

Second, we are interested in investigating the unique properties of databases of local image features compared to "general" nearest neighbor search problems, and which impact these properties have on the design choices for a scalable retrieval method.

The main contributions of this chapter are: 1) an extensive evaluation of the "classic" algorithms LSH, metric trees, and the more recent Redundant Bit Vectors (RBV) in terms of NN search versus scalability on large databases of local image features. 2) An evaluation of strategies to move from NN search to an image or object retrieval system. 3) An exhaustive evaluation and comparison to recent clustering-based methods for large benchmark datasets. We limit our evaluation to the appearance feature indexing problem in this chapter. Geometric verification can be added to improve any such method.

The chapter is organized as follows: we start with an introduction of datasets and evaluation measures in Section 6.2. We continue with summaries of the methods we consider for evaluation in Section 6.3. This is followed by an evaluation in terms of nearest neighbor (NN) retrieval performance in Section 6.4. Section 6.5 discusses the steps to get from NN search to object retrieval. In Section 6.6 we evaluate a selected algorithm (forests of metric trees) on large datasets and compare to other state-of-the-art methods. The chapter ends with a discussion of related work and our findings in Sections 6.7 and 6.8.

This chapter is based on an evaluation carried out by my two former students David Scheiner and Reto Schwarz during their Masters thesis [Scheiner and Schwarz, 2007].

6.2 Datasets, Features, and Evaluation Metrics

We will evaluate the different methods on recent benchmark data from [Nistér and Stewénius, 2006; Philbin *et al.*, 2007] and on additional large datasets, which we collected ourselves. More specifically, throughout the chapter we will report results on the following sets:

UK Set: A collection of 2500 objects, each shot from four different viewpoints, introduced in [Nistér and Stewénius, 2006].

UK Set Small: The first 5 000 images of the UK Set. The query set consists of the first viewpoint of the first 500 objects. All query images are removed from the database, resulting in a data set of 4 500 images.

Oxford Set: A dataset containing 5000 images of different Oxford landmarks from [Philbin *et al.*, 2007].

Dataset	# Images	# Features	# Q	# Q-feat.
UK	10 200	15 297 858	2550	3 863 723
UK S	4 500	1 736 564	500	167 617
Oxford	5 063	21 406 572	55	105 881
Amazon	52 002	55 021 426	79	400 559
DMOZ	1 146 819	328 528 420	-	-

***Table 6.1**: Dataset Statistics: Number of database images, number of database features, number of query images and query features.*

Amazon Set: A database of 52 000 DVD covers we downloaded from amazon.com. 79 query images were photographed from real DVDs using mobile phone cameras.

DMOZ Noise set: 1 million random images we downloaded following links on the first few levels of the Open Directory Project. This set is intended as a large noise set.

Figure 6.1 shows some example images from the datasets, and Table 6.1 shows a summary of the dataset sizes. For each set we extracted SURF [Bay *et al.*, 2006b] features, which results in a bag of 64-dimensional feature vectors for each image. We chose SURF features due to their fast extraction times and good recognition performance in prior evaluations.

We consider several evaluation measures. To investigate the quality of nearest neighbor (NN) search, we measure how well an algorithm performs in finding the true NN. The effective distance error E was proposed by [Gionis *et al.*, 1999]: given the distance d_t to the true NN and the distance d_{alg} to the NN found by the algorithm for each query point $q \in Q$, the effective distance error is calculated as follows:

$$E = \frac{1}{|Q|} \sum_{q \in Q} \left(\frac{d_{alg}(q)}{d_t(q)} - 1 \right) \tag{6.1}$$

where Q is the set of all query points. It measures the average error resulting from the approximate nature of the algorithms. In addition to E, we also consider the fraction of true NNs found (for a set of ground truth query points) by each algorithm.

To evaluate algorithms on the retrieval system level, we use the mean average precision (mAP) measure, as proposed by [Philbin *et al.*, 2007]. Average precision (AP) is the area under the precision-recall curve for a query. An ideal precision-recall curve has a precision of 1 over all recall levels and with this an average precision of 1. Mean average precision is obtained by averaging the AP values for several queries of a test set. On the UK Set the same metric as in [Nistér and Stewénius, 2006] was used, which is a score defined by the average fraction of correct (*i.e.* depicting the same object) images in the first four results, *i.e.* a score in the range $[0, 4]$.

***Figure 6.1**: Query image and three true positives for each the UK Set, Oxford Set and Amazon Set*

6.3 Overview of Methods

We complement the works using k-means clustering by investigating several NN search methods; Locality Sensitive Hashing (LSH) [Datar *et al.*, 2004], Redundant Bit Vectors (RBV) [Goldstein *et al.*, 2005], and metric trees [Uhlmann, 1991]. All methods have been suggested to perform well in high-dimensional spaces. LSH is probably the most popular approximate NN search technique today. Metric trees (or the similar balltrees) represent "classic" exact methods with good performance. Notably, in [Liu *et al.*, 2004] a study on smaller datasets showed that metric trees can be adapted to handle the approximate NN problem with a speed-up of up to 30 times over LSH. RBVs are a rather new approach which recently received interest due to their speed-up over LSH at low memory consumption. These methods represent a good variety of different approaches to NN search with a good performance in earlier studies and with a high potential for scalability. In the following sections, we evaluate how the methods compare in terms of finding the correct NNs versus their consumption of resources. We start with a summary of the theory for each method.

6.3.1 Locality Sensitive Hashing

LSH [Gionis *et al.*, 1999] is a popular family of algorithms for approximate NN search. Its basic idea is to apply several hash functions to the points in the database, which ensure that points lying close to each other have a higher probability of collision than points far apart. A query point is treated with the same hash functions, and points found in the matching buckets are retrieved. We chose an algorithm proposed in [Datar *et al.*, 2004], which works directly in the Euclidean space (unlike [Gionis *et al.*, 1999] and related methods which operate in Hamming space). The employed hashing procedure maps d-dimensional points $\mathbf{d} \in \mathcal{R}^d$ to the integer range by random projections

$$h(\mathbf{d}) = \lfloor \frac{\mathbf{a} \cdot \mathbf{d} + b}{w} \rfloor \tag{6.2}$$

With this, the integer range is segmented into r sections of width $w = \text{INT\_MAX}/r$. b is randomly selected from a uniform distribution $[0, w]$. The elements of vector $\mathbf{a}$ are drawn from a $p-$stable distribution, in our implementation a Gaussian distribution, since we use an L_2 norm (see [Datar *et al.*, 2004] for details). For increasing r, the width w of the segment in which random projections fall decreases. By this scheme, close-by points are mapped into the same sections of a projection with high probability. Random collisions are further minimized by concatenating k random projections

$$\mathbf{x}(\mathbf{d}) = (\lfloor \frac{\mathbf{a_1} \cdot \mathbf{d} + b}{w} \rfloor, ..., \lfloor \frac{\mathbf{a_k} \cdot \mathbf{d} + b}{w} \rfloor) \tag{6.3}$$

into a random has function $\mathbf{x}$. By increasing k, the probability that two points far away accidentally map to the same $\mathbf{x}$ diminishes, while two close points likely result in the same $\mathbf{x}$-value. Therefore the probability that two random points map to the same hash key converges to zero.

6.3.2 Redundant Bit Vectors

Redundant Bit Vectors (RBVs) are a rather new method, proposed in [Goldstein *et al.*, 2005] as an approach for high-dimensional nearest neighbor search and originally intended for indexing audio fingerprints. While one of the main strengths of the method is its ability to quickly discard items that don't have a match in the database ("negative queries"), it can also handle (bounded) NN search. [Goldstein *et al.*, 2005] reports good results, especially an order of magnitude speedup over LSH in an audio fingerprinting database. A particularly interesting property of RBVs is their small memory footprint.

The main idea of the RBV algorithm is to quantize the query space (instead of the feature space, as in almost all other fast NN matching approaches). The argumentation is, that in high dimensional spaces locality properties are weak, and thus the grouping of the database into bins defined by locality is rarely helpful while searching. To that end, each dimension of the query vector is split into Q bins. Next the algorithm constructs an individual hypercube of side length 2ϵ around each data point $x_1 \dots x_n \in \mathbb{R}^d$. This hypercube overlaps with one or more of the Q bins in each dimension. The set of all n data points falling into a bin is represented by a bit vector of length n, where a 1 means that the corresponding point extends into the bin for that dimension.

Thus for each dimension, a $Q \times n$ bit field representation of the data is created. The total memory requirement for all bit fields is $Q \cdot n \cdot d$ bits. The side length $2\epsilon_i$ of each hypercube is determined by calculating the average distance to t randomly selected data points. The resulting value is then multiplied by a tunable factor r.

Note that the size of the Q bins for each dimension depends on ϵ and that the bins are not of equal size. To determine the sizes of the bins, tentative bin boundaries are created by adding and subtracting ϵ values from each data point. For each dimension, a list is built and sorted. The lists are partitioned in $2n/Q$ blocks. The first and the last element in a block define the boundaries of a bin.

At query time, for all dimensions the bins containing the query point are selected, and the associated bit vector column is retrieved. All bit vector columns found are combined using the bitwise `AND` operation on blocks of 64 bits (64 bit architecture). The i-th non-zero entry in the resulting vector indicates that the query point falls within the hypercube of the i-th data point. For a small r, this often leads to a result vector containing only zeros, since the algorithm was designed to discard negative queries. For (approximate) NN retrieval a slightly adapted version is required. We introduce the following cut-off criterion: while we `AND` the dimensions, naturally with each `AND`-operation less non-zero bit vector elements are left. If the number of non zero entries drops below a threshold value, we stop early. This leaves us with a set of NN candidates instead of only one single closest, which we search in linear fashion. This procedure adds softness to the method, which helped finding more true NN at low cost in speed. In fact, by stopping `AND`-ing early, we save some processing time, which is "recycled" for the linear search at the end.

6.3.3 Metric Trees

Metric trees were introduced in [Uhlmann, 1991]. In a metric tree, the data points are partitioned by their distance to certain pivot points. At every node of the tree, two points (the pivots) of the dataset are selected. Then the distance to the two pivots is calculated for all points in the node, and the points are assigned to the

closer node. In this fashion the data is split into two sets, which are sent to the child nodes, and the process is repeated. When a node only contains one data point, the process is stopped. This node then constitutes a leaf of the metric tree.

Literature on metric trees describes many possibilities to optimize the tradeoff between quality and resource-consumption of these tree algorithms [Ciaccia *et al.*, 1997; Liu *et al.*, 2004; Moore, 2000]. However, since most works only deal with comparatively small data sets (up to 100,000 data points) or "low" dimensionality (up to $d \leq 30$), it is not immediately evident how these optimizations would perform on large data sets. Thus, we implemented several variations of metric trees and tested them on our data. The methods differ by three characteristics: the partitioning scheme, the handling of data near partition boundaries, and pivot selection.

Partitioning schemes are either spherical or symmetrical. For the former, a hypersphere is drawn around one of the pivots. All data points inside the sphere are assigned to the node whose pivot defines the sphere's center. All remaining data points are assigned to the other node. In a symmetrical partitioning scheme, a hyperplane is created which separates the space at equal distance to the pivots.

Since splitting the data between the pivots is a hard decision, points near the boundary might lead to false decisions while traversing a tree. One method to avoid this problem is to introduce overlapping boundaries, or "spilling" [Liu *et al.*, 2004; 2007]. Points lying inside an overlap region belong to both pivots. Therefore, these data points are duplicated and assigned to both child nodes. This procedure can be carried out while building the tree ("buildup spilling") or during lookup ("lookup spilling", or "backtracking").

Finally, there are several ways to select the pivots, the simplest being a random selection. We use a "ping-pong" scheme: at each node one pivot is selected at random, its distance to all other points in the current set is calculated, and the point farthest away is selected as the second pivot. This process is repeated until no points remain. After intensive prior experiments, we identified the following variants of metric trees as promising candidates:

mtreesph: tree with spherical partitioning. The radius of the sphere is the mean of the distance from the pivot to all other datapoints. Lookup spilling is implemented using an overlapping region, defined as the fraction d of the sphere radius. If a query point lies inside the overlap range, it is "duplicated" and sent to both children.

mtreesym: symmetrical partitioning and lookup spilling by an overlap region along the separating hyperplane instead of the sphere, defined by the fraction d of the distance between the pivots.

Note that spilling requires additional memory for the duplication of data points. One way to optimize memory usage is the hybrid spill tree [Liu, 2006]. The basic idea is not to spill at each node, but only where it would make a substantial difference. We tried several proposed schemes but did not observe improvements over regular spilling on our data.

6.4 Evaluation in terms of NN Search

Our first set of experiments compares the three proposed algorithms in terms of NN search. A good algorithm has the following desirable properties:

1. It is fast in finding the NN.
2. The index representation is compact (in memory).
3. The time to build the index is manageable.
4. There are few parameters to optimize.

Thus, in this series of experiments we set the quality of NN search in relation to the first three criteria. The last criterion (number of parameters) will be discussed at the end of this section. We compare the different NN methods on the *UK Small* dataset (see Section 6.2). For each algorithm, we varied its parameters and studied the effect on performance in precision, resource consumption, and build time. We selected the result which showed the best performance and/or performance vs. resource trade-off and compare them in Figures 6.2 and 6.3. Figure 6.2 shows the results for the effective distance error measure. The curves are obtained by varying one free parameter for each algorithm.

For LSH, we varied the parameter r (number of sections) between 150 and 375. Small r values result in a large number of collisions. Therefore, each bucket contains many data points, which have to be searched in a linear fashion. In general, this leads to a low error rate at a higher cost in computation. The number of concatenated functions k is set to 10, as suggested by [Datar *et al.*, 2004]. Other values were also tested but do not push the results beyond those reached with $k = 10$. Increasing k leads to fewer collisions, whereas decreasing r leads to more collisions. As both parameters have a coupled effect, only r is swept. The hash table size was set to 12 bits. Different table sizes were tested, but did not result in an improvement.

For RBV, the parameter r (which controls the hypercube size) was varied in the range $[0.05, 0.3]$. Several values for the number of bins per dimension Q were tested. $Q = 12$ gave the best results and is shown in this summary. As suggested in [Goldstein *et al.*, 2005], t (number of randomly selected datapoints) was set to 100.

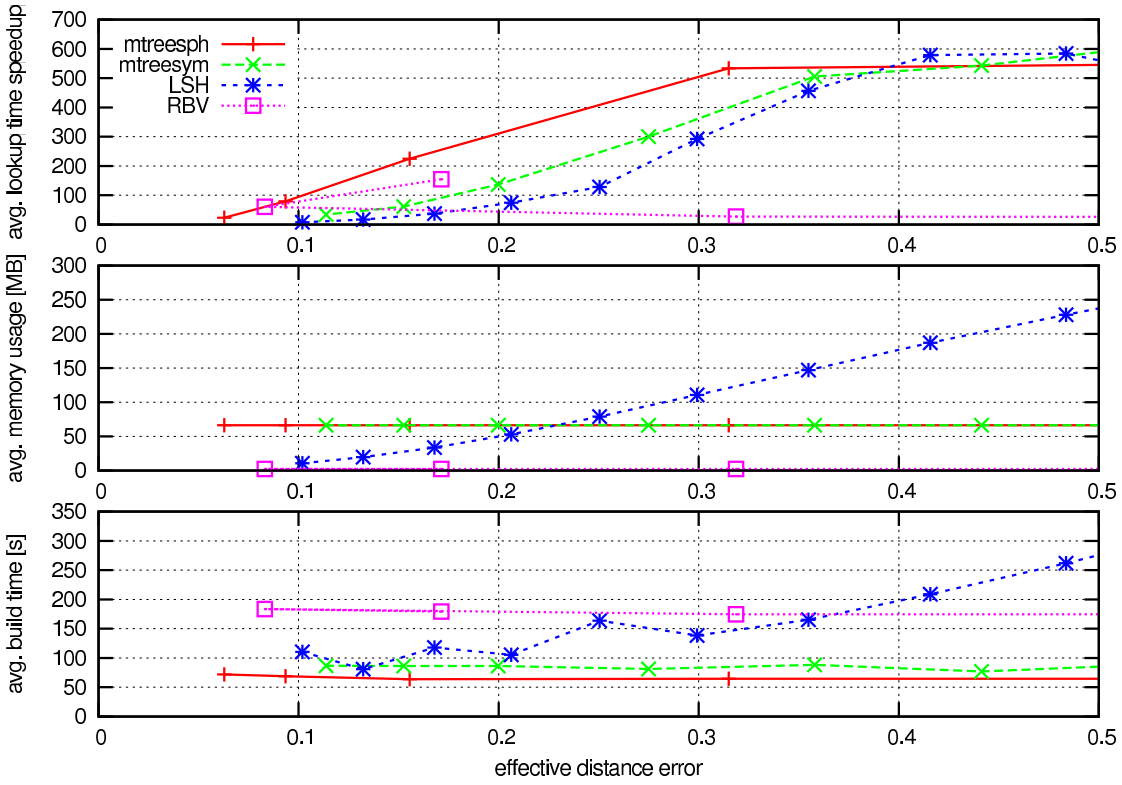


***Figure 6.2**: Quality of NN search: Effective Distance Error*

Finally, we show two metric tree versions using spilling during lookup, which gave the best results in our prior evaluations. Curves are generated by varying the spilling parameter d between $[0, 0.1]$ for the spherical mtree *mtreesph* and $[0, 0.7]$ for the symmetrical mtree *mtreesym*. Higher overlap d results in better error rates, but comes at a cost in speed, since more branches have to be considered during lookup.

As Figure 6.2 shows, all algorithms reach about the same level of effective distance error, with metrics trees and RBV slightly better than LSH. The highest speed-up over linear search at low effective distance error is reached by the trees (first row of plots). It is interesting to observe how LSH can be increased in speed, but not only at cost in error, but also at a cost in memory requirements and build time (2^{nd} and 3^{rd} rows). The second row also shows, that the lowest memory usage is achieved by RBVs. The memory consumption of trees is higher than the one of RBV, but stable compared to LSH. Trees also exhibit the lowest build time (3^{rd} row). Also note the low absolute value of the build time: in about 1 minute, we can build a tree for the *UK Small* dataset, which consists of 1.7 million features.

Figure 6.3 shows the results for the fraction of true NNs found. The curves are again obtained by sweeping through the same parameter ranges described above. RBVs recover the largest fraction of true NNs. The worst performance in these terms is

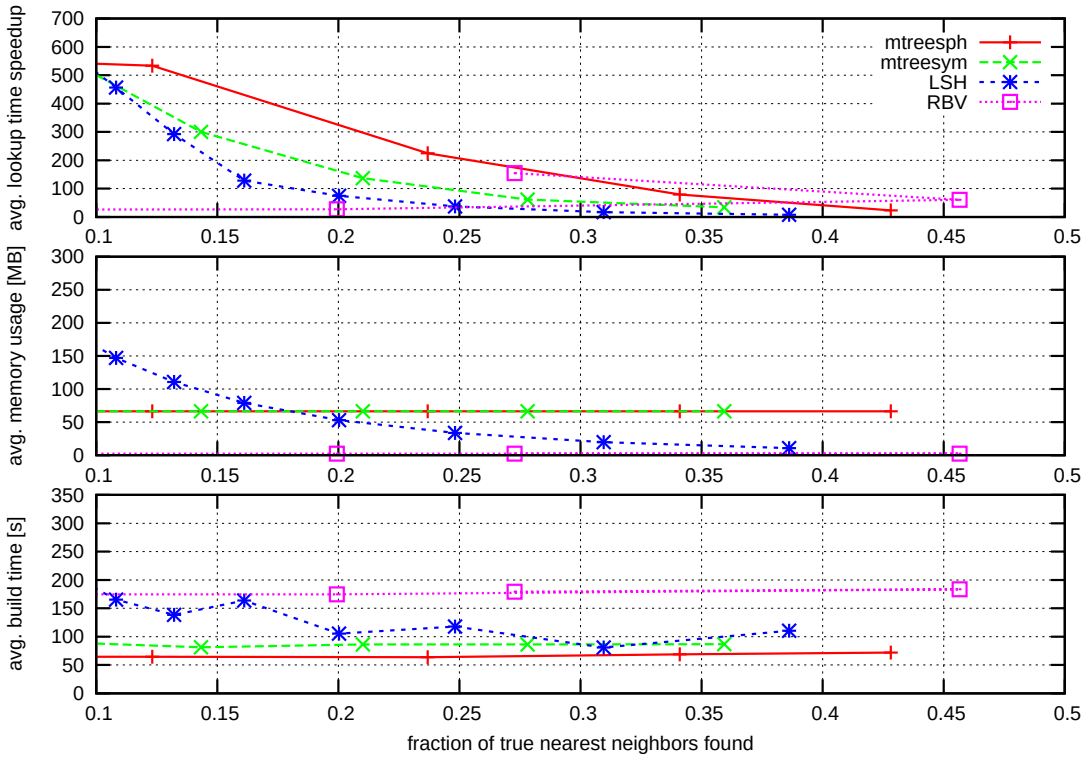


***Figure 6.3**: Quality of NN search: Fraction of True Nearest Neighbors*

again achieved with LSH. Trees recall a smaller fraction of true NNs, but offer the highest speed-up at low to medium NN-recall values.

With our intention of scaling to even larger datasets, in summary the best results are achieved using the metric trees. They offer a good trade-off in scalability (speed-up and build time) at reasonable loss in precision over linear search. RBV performs very well in terms of precision. However the speed-up over linear search is not high enough. Considering that it is a fairly novel method, further research might lead to improvements here. Surprisingly, LSH did not perform as well as expected and seems to be the worst of the three methods. In spite of our careful implementation, a possible explanation might be that we did not find the optimal parameters for this method. However, even if this is true, it does not speak in favor of LSH. On very large datasets it is extremely time-consuming — if not impossible — to optimize many parameters.

Note that by using local features, we can afford losing some of the NNs: objects in images are typically covered by hundreds of features. It just has to be assured that a sufficient number of features is extracted, so that later processing stages (such as RANSAC) have enough data, even if we potentially lose a large fraction of NNs. This situation is different from early research in the field of large-scale image databases:

the global image features typically used in those early works did not offer this kind of redundancy.

6.5 From NNs to retrieval in large databases

From the experiments in the previous section, we learned that we can achieve significant speed-up over linear search, mostly at the cost of losing some true NNs. In this section, we want to devise strategies to handle those losses and recover some of the precision. The first question to be answered is whether a retrieval system requires all the true NNs to achieve good overall precision. Figure 6.4 (top) shows overall recognition score[1] versus fraction of true NNs found in the *UK Small Set*. Linear search is at the top-left with the baseline score of 1.62. It is clearly visible that all algorithms come close to the performance of linear search by recovering only $30\% - 50\%$ of the true NNs. This effect is due to the aforementioned redundancy coming with the use of local features, which liberates us from retrieving the true NN for each feature. An immediate conclusion from this insight is that we might try to recover some of the true NNs by retrieving k near neighbors instead of only one — hoping that by this we add less noise to the results than we gain precision.

Thus, in Figure 6.4 (middle) we show an experiment which reports mAP vs. k for the *Oxford Set* using linear search over the entire set. The ranking of results is defined by the number of "votes" each image gets, *i.e.* by the number of features from the query image that matched in the database image. The scatter points correspond to different parameter settings of each algorithm. Since we retrieve not only the NN, but k near neighbors, we can also apply different voting strategies, which correspond to the different curves. Regardless of the voting strategy, a maximum performance is reached around $k = 15$. The gain over using only the closest NN is quite substantial. The three different voting strategies are intended to suppress noisy votes and are defined as follows: *singlecount* allows only one vote (match) for each feature per image, *inverse weighting* and *inverse quadratic weighting* weight votes by the inverse distance between data and query point. While the inverse weighting even lowered the results, *singlecount* weighting helped to gain a few percent. This effect can be explained by the removal of multiple votes stemming from repeated and non-discriminative patterns.

[1]Here, a slightly modified UK score was used. Query images are not included in the result set, and the remaining 3 correct images are weighted by their retrieved position, the maximum is $1 + 1/2 + 1/3 = 1.833$.

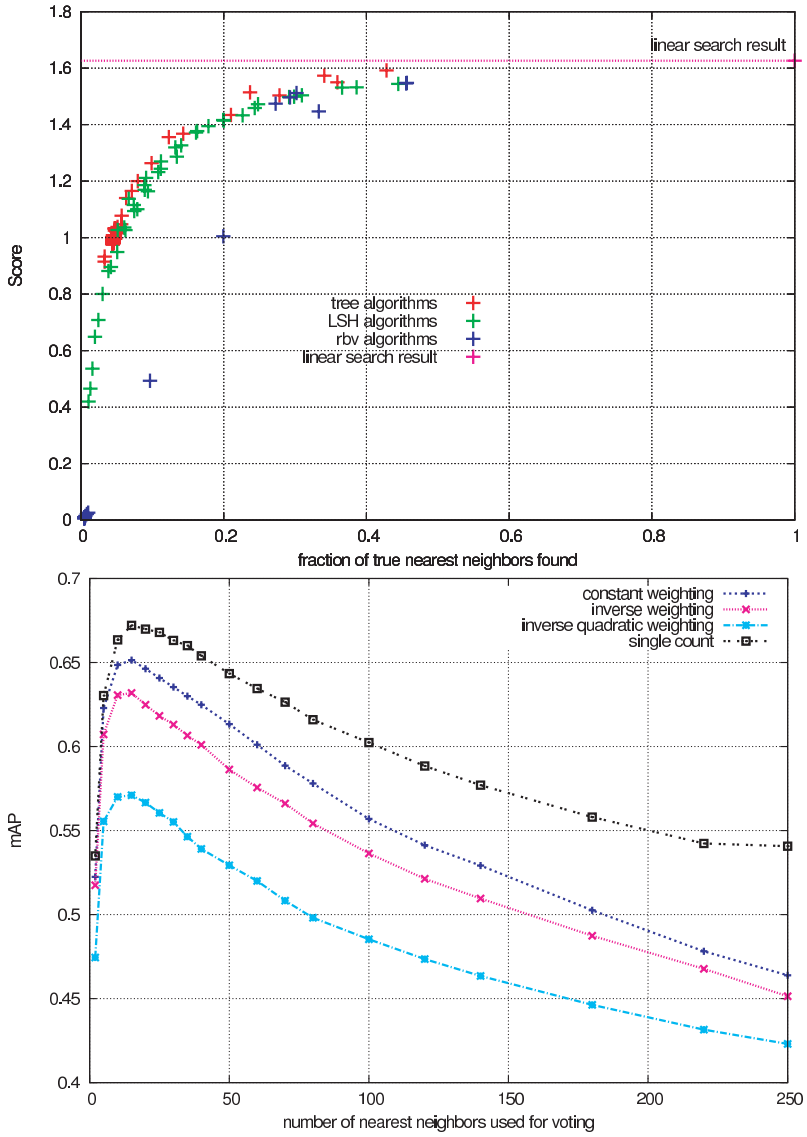


Figure 6.4: *True NN vs. ranking score (top) and mAP vs. number of near neighbors k (bottom)*

6.5.1 Forests of randomized metric trees

Based on the experiments in Section 6.4 and the results above, we focus on one of the algorithms to scale to large datasets. We chose the metric tree due to the speed-up we observed in Section 6.4 and the ease to handle it compared to the other methods. The only drawback is the larger memory consumption. Considering, however, the lesson learned from Figure 6.4 that it is not only sufficient but beneficial to retrieve k near neighbors, we do not have to build complete trees. Rather, we can build a smaller tree from a random sample of the data. All remaining points of the set are then inserted into the tree and appended to a list of features for the leaf they fall in. This way, we save memory and are able to retrieve multiple close matches for query points at the same time. In fact, this representation is very similar to the "visual words" obtained from k-means clustering in [Nistér and Stewénius, 2006; Philbin *et al.*, 2007]. Note that metric trees are in general unbalanced. The number of levels on each side of the root is data dependent, trees will generally be deeper where data is dense.

During lookup, each query point is inserted into the tree, and all the images in the matching leaf receive a vote. In addition, lookup-spilling is applied, such that each query point may fall in multiple leaves, *i.e.* may generate multiple votes. Furthermore, we introduce a new spilling variant, which we call "insert spilling": while inserting the full data in the tree (which was built from a random subset of the data) we have this additional possibility for spilling.

Finally, instead of using one single tree, we can use multiple smaller trees, each built from a random subset of the data, *i.e. forests of randomized metric trees*. In the next section we evaluate this method on a variety of large datasets. In spirit this idea is along the lines of [Moosmann *et al.*, 2006; Ozuysal *et al.*, 2007], where different types of randomized trees have been used. The differences here are that randomness is based on the random subset used to build the trees, that our trees are built on NN search, and that we consider retrieval in very large datasets (as opposed to classification on smaller sets).

6.6 Evaluation on Large Datasets

In the previous sections we compared several approximate NN search algorithms and concluded that metric trees offer the best opportunities to scale to the truly large datasets introduced in Section 6.2. Below, we will evaluate the metric trees under this aspect, and compare their performance to the k-means methods of [Nistér and Stewénius, 2006; Philbin *et al.*, 2007].

We first compare performances on the Oxford data using the mAP measure for different tree configurations. The trees we compare differ in the size of the random

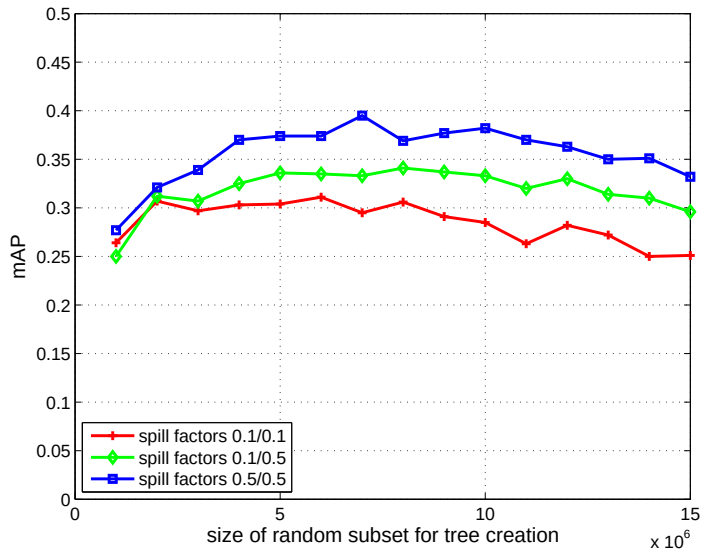


***Figure 6.5**: Forests of metric trees: mAP for different random sample sizes and spilling settings.*

data sample used to create the tree, in the spilling parameters, and, if multiple trees are used, in the number of trees in the forest. All trees are symmetric metric trees. Figure 6.5 shows results for individual trees created from random samples of different sizes and with different spilling configurations. One can see that the performance is dependent on the sample size and its optimum is reached at a fraction of the full set size. This is the same effect observed before for voting with k near neighbors instead just the NN: if the tree size approaches the size of the full dataset, only few features remain in the leaf nodes and few votes are generated for each query feature.

The three curves in the plot represent three different spilling configurations, each defined by a pair of spill factors. The first spilling parameter denotes *insert spilling*, the second parameter is for *lookup spilling*. A higher spilling factor (0.5 vs. 0.1) gives better results – both for spilling during insert and lookup. Note how the effect of spilling appears only for sample sizes larger than about 2M.

Table 6.2(a) shows mAP results for different forests. Using larger forests improves the mAP up to 0.539. Again, increasing spilling from 0.1 to 0.6 improves the recognition rates (0.444 vs. 0.539). We also tried spilling values higher than 0.6. However, the additional gain in precision was small, while the lookup time increased substantially. Table 6.2(b) compares these results to the results reported in [Philbin *et al.*, 2007] on the Oxford Set. Note that this comparison is in terms of a "bag of words model", before verification using multiple-view geometry. These results are competitive with the state-of-the-art and lie in between hierarchical k-means (HKM) [Nistér and Stewénius, 2006] and approximate k-means (AKM) [Philbin *et*

(a)

Forest	mAP		Forest	mAP		
	.1/.5	.5/.5		.1/.5	.5/.5	.6/.6
1 × 3M	0.307	0.339	1 × 7M	0.333	0.395	0.444
2 × 3M	0.343	0.380	2 × 7M	0.378	0.400	0.485
4 × 3M	0.380	0.412	4 × 7M	0.419	0.438	0.525
6 × 3M	0.391	**0.418**	6 × 7M	0.444	0.463	**0.539**

(b)

Method	Oxford [mAP]	UK [Score]
HKM-1 [Nistér and Stewénius, 2006; Philbin *et al.*, 2007]	0.439	3.16
HKM-4 [Nistér and Stewénius, 2006; Philbin *et al.*, 2007]	0.353	3.29
AKM [Philbin *et al.*, 2007]	0.618	3.45
This Work	0.539	3.34
Linear Search	0.672	3.53

(c)

Forest	% 1^{st} cor.
1 × 1M	46.8%
1 × 5M	49.4%
6 × 1M	70.8%
6 × 5M	83.5%
Linear	95.0%

***Table 6.2**: Evaluation of forests on Oxford set (a), comparison to other works on Oxford and UK sets (b), results on Amazon set (c).*

al., 2007]. Note that our best single tree result on the Oxford set (Table 6.2(a)) is higher than both HKM versions. As a baseline, we also report the result we obtained using linear search. This result is still slightly better than all three methods, confirming that linear search constitutes indeed a suitable baseline – which validates our assumptions from Section 6.4. The rightmost column of Table 6.2(b) compares performance on the (full) UK set. Here, metric trees perform again better than [Nistér and Stewénius, 2006] and only slightly worse than [Philbin *et al.*, 2007].

With the Amazon set we introduce a new set and a new task. It is similar to the CD retrieval application in [Nistér and Stewénius, 2006]. We envision a scenario where users take picture of a DVD cover using the camera of their mobile phone and after recognition may read critics about the DVD, buy it on-line etc. This is different from the task on the Oxford set presented in [Philbin *et al.*, 2007], where retrieval is performed only for a query selected from images already in the database, *i.e.* the feature vectors of the query images were in the dataset while the visual vocabulary was created. Further, the task is to retrieve a correct item in the first

(a)

Set	Tree	Time	MB	# feat DB	# feat./q
Oxford	1M	0.09s	80	21 406 572	1925
	5M	0.12s	400		
	10M	0.15s	800		
UK	1M	0.08s	80	11 434 135	1515
	5M	0.12s	400		
	10M	0.17s	800		
Amazon	1M	0.08s	80	55 021 426	1370
	5M	0.09s	400		
	10M	0.10s	800		

(b)

Set	Tree	t/tree	mAP	$\%1^{st}$ corr.
Oxford+DMOZ	5x3M	0.15s	0.215	-
UK+DMOZ	10x1M	0.14s	0.230	-
Amz.+DMOZ	10x1M	0.08s	-	44.3%

***Table 6.3**: Lookup times and memory usage in MB for single trees (a), performance metrics in DMOZ set (1 million images)(b).*

position. Query images were taken with mobile phone cameras, under challenging viewpoints, with specularities *etc.*, and are very challenging compared to the data in [Nistér and Stewénius, 2006; Philbin *et al.*, 2007]. The database consists of about 52 000 covers with over 55 million features. Table 6.2(c) summarizes the results. The spilling setup was 0.1/0.1 for all configurations. With a forest of 6 trees built from 5 million sampled features, we reach a precision of 83% in the first position of the ranked list. This is quite good, considering the difficult query images and the "one shot" option for this task. Some example results for this task are shown in Appendix A.

6.6.1 Computation Times and Scaling

We first report lookup times for the Oxford, UK, and Amazon sets in Table 6.3(a). It shows the average lookup time per image using a single tree of the given size. We also report the size of the database and the average number of features per query image. To test the scalability of the tree method, we performed tests on the DMOZ noise set with over one million images. The images from the original datasets were mixed into this set before tree construction, and the same retrieval experiments were performed again. Note that the data from DMOZ is extremely challenging, since it contains truly random images, some of them with an excessive number of features, which influence the "bag-of-words" voting procedure substantially. Recognition rates and runtimes on this set are reported in Table 6.3(b). The effect of the noise set reduces

recognition performance by about 50%. This is similar to the results in [Philbin *et al.*, 2007], where a reduction by about 40% was reported for a noise set consisting of images retrieved from flickr. We assume that the higher loss here can be explained by the more "distractive" content of the DMOZ set. In terms of speed, the metric trees scale very well with a lookup time of about 0.1s per tree for over 300 million features. In the worst case, where a forest of trees would have to be searched sequentially, we would achieve retrieval in 1 second on 1 M images using a single PC. For many practical applications, the tree search would however be parallelized.

Memory usage is influenced by several parameters. Each node requires: two unsigned integers (each 4 bytes) to indicate the array position of the actual descriptor data for the pivot. (No pointers are used since they consume 8 bytes on a 64-bit architecture). Another two unsigned integers store the ids of the child nodes. Therefore, the size of one node is $n_s = 16\ bytes$. A SURF descriptor vector can be described with $d_s = 64$ bytes. The memory required for the tree is

$$\text{mem}_{\text{tree}} = i_n \cdot (n_s + d_s) \tag{6.4}$$

where i_n is the size of the random sample used to build the tree. With this, a tree built from 1 million vectors requires 80MB using byte-valued SURF features. The 4th column of Table 6.3 shows the memory usage for the different tree examples. The complete $6 \times 7M$ forest for the best result obtained on the Oxford set would require 3.36GB.

For the inverted files in the leaf nodes, storage depends strongly on the implementation, a straightforward approach with a leaf size l_s (typically 4 bytes) and a total of d_n features in the database requires

$$\text{mem}_{\text{leaf list}} = d_n \cdot l_s \tag{6.5}$$

For a large dataset such as the Amazon or DMOZ set, this would add to the memory usage 220MB, or 1.2GB respectively. If insert spilling is used, these numbers increase, since some points fall into multiple leaves. However, there is an extensive body of work from document retrieval on how to store the lists more efficiently using several kinds of compression. A good summary can be found in [Zobel and Moffat, 2006].

6.7 Related Work

Searching large databases of local visual features is a topic, which is receiving increasing attention. The main body of work for that particular application has only been carried out in the last few years, following the increased popularity of local visual features for object recognition. Early works, such as [Lowe, 2004] proposed

to use classic *k-d* trees coupled with the best-bin-first method and demonstrated scalability on 100 000 SIFT descriptors. The well-known work [Sivic and Zisserman, 2003] by Sivic *et al.* suggested video retrieval based on clustering appearance features into "visual words" using k-means. Finding nearest neighbors for query features is replaced by finding the closest cluster centroid (visual word), and indexing is solved using inverted files of visual words. The rather simple k-means clustering method seems to adapt to the structure of the feature space surprisingly well and results in outstanding retrieval results on full feature movies. The main disadvantage of such an approach is the scalability of the clustering process, especially for large numbers of visual words or clusters. Thus, [Nistér and Stewénius, 2006] recently suggested a hierarchical k-means approach and [Philbin *et al.*, 2007] proposed an approximate "flat" k-means. Both methods speed-up the k-means clustering process significantly and therefore allow creating larger vocabularies. Both works also show significantly improved recognition performance using larger vocabularies.

In addition to [Nistér and Stewénius, 2006; Philbin *et al.*, 2007; Sivic and Zisserman, 2003] our evaluation relates to a large body of work dealing with (approximate) nearest neighbor retrieval in high-dimensional vector spaces, independent of the specific application or data-type. Not astonishingly, well-known approximate nearest neighbor search methods such as *k-d* trees, metric trees and Locality Sensitive Hashing (LSH) have been compared on several smaller datasets before, for instance in [Liu *et al.*, 2004]. However, these comparative evaluations have not been carried out on databases of local image features, the characteristics of which have a strong influence on design choices, as we could show in Section 6.5 of this chapter.

6.8 Discussion and Conclusions

In this chapter, we have evaluated methods for large-scale retrieval in databases of local image features at several levels of the processing pipeline. Our first contribution is the evaluation of some of the most popular (approximate) NN methods on current benchmark data for large-scale object recognition. Our comparison of the methods LSH, RBVs and metric trees has shown that among those methods, metric trees offer the best speed-up. For the comparison between LSH and metric trees this confirms results in [Liu *et al.*, 2004] on smaller datasets. While [Goldstein *et al.*, 2005] showed that RBVs outperform LSH, we could show that at least for local image features, spilling metric trees outperform both LSH and RBV. The cost for this speed-up comes in a loss of true NNs retrieved, compared to linear NN search.

We thus evaluated the influence of this loss on the overall recognition performance of a retrieval system with local image features. We could show that close to linear-search performance can be reached with only $30 - 50\%$ of true NNs found. Furthermore, we demonstrated that retrieving k near neighbors (instead of the true NN)

improves results further. We also evaluated different match-voting strategies to rank images based on the matches between their local features. In our experiments, we observed an optimum at $k = 15$ combined with *singlecount* voting. These findings probably hold for most systems with local features. With this we demonstrated that due to the redundancy offered by local features, choosing a less precise (in terms of true NN) but faster method is beneficial for Internet-scale retrieval systems, since the loss in precision can be absorbed with appropriate strategies on higher levels on the system.

It is somewhat astonishing, that *LSH* did not perform as well as expected. One reason which canot be excluded, that our implementation is not as optimized as the one used by the authors of the original work. In any case however, LSH is a method with many parameters to optimize, which makes evaluation on large datasets difficult.

Due to their superior performance, metric trees were further evaluated and combined to forests built from random subsets of the data. We showed that spilling helps improve precision, and we introduced "insert spilling" for trees built from sampled data. The results were validated across the largest currently available benchmark data sets and on a new set of 50′000 DVD covers from amazon.com. Simulating the data available on the Internet, a large-noise set of 1 million images from DMOZ was used to test robustness against noise. The overall recognition rates of spilling metric trees compete with recent clustering methods [Nistér and Stewénius, 2006; Philbin *et al.*, 2007] on benchmark data, while offering substantially faster index build times not relying on an iterative clustering procedure.

7

Conclusions and Outlook

In this thesis we have investigated mining and retrieval in databases of visual data at the object level. Building on state of the art local appearance methods for object recognition, our main contributions are in the fields of mining feature configurations as representatives for object classes, multimodal mining of objects and events from community photo collections, and multimodal retrieval application for mobile phones. Further contributions include a method for detection of text in natural scenes and an evaluation of algorithms for scalable retrieval in databases of local features.

7.1 Contributions

The main contributions of this thesis can be summarized as follows:

In Chapter 3 we applied itemset mining algorithms in the domain of visual data. We adapted this simple, but effective class of methods to work with configurations of local visual features. We demonstrated, ho the spatial arrangement of visual words in semi-local neighborhoods can be encoded as transactions and subsequently mined to identify repeating patterns of local feature configurations in the data. We showed, how the detected patterns can be used to mine specific objects from video data. It turned out to be helpful to base the creation of candidate neighborhoods on motion segmentation. The same approach was extended to mine feature configurations as evidence for the presence of instances of object classes. We could show that the mining algorithms can be used to solve the task of learning frequent feature configurations, relevant for a given object class. Conducting experiments on state of the art benchmark data, we could demonstrate that the mined configurations show better evidence for the presence of object class instances than single visual words. Using the of the mined configurations in the implicit shape model [Leibe *et al.*, 2008], however, did not show the expected improvement compared to single features. In summary, in spite of their simplicity, the itemset mining methods turned out to

be suitable tools for the tasks of mining visual data. Compared to other methods in the field of object class recognition, itemset mining will probably play out its strengths only when applied to larger amounts of data, where a rough, but efficient localization of candidates is required.

In Chapter 4 we took mining from the feature level to the object level. We introduced a combination of methods, which allows for mining objects and events from community photo collections on the Internet. The approach relies on geotagged photos, which are clustered based on their similarities calculated from local feature matches. Beyond the visual cues we extended our mining method to include multimodal cues such as the textual tags describing the photos. A classification based on the meta-data of the clustered photos was used to divide clusters into objects and events. Textual labels were learnt for the clusters efficiently, by using frequent itemset mining in order to identify combinations of tags relevant for the cluster's contents. These textual labels were also used to crawl possibly relevant Wikipedia articles from the Internet. Closing the loop to the visual modality, images in the crawled articles were matched back to the mined clusters, to verify the hypothesized assignment between article and photo cluster. Finally, we demonstrated, how the mined photos can be used to derive object-level auto-annotations of objects such as landmark buildings in holiday snaps. Experiments were conducted on hundreds of thousand of photos downloaded from the Internet. Annotation quality on the mined data was evaluated on manually labeled groundtruth of several hundred images. The recognition rates reached with 70% a very satisfactory level on this challenging task.

In Chapter 5 we took an application-centric view of object recognition. We demonstrated several prototypes for object recognition applications, with a special focus on mobile devices. Several prototype implementations for mobile visual search on the object-level were discussed and compared. In all cases the object recognition is performed on the server-side, while client applications display the results. Namely, we compared user-initiated recognition, real-time object recognition from streaming video between client and server, and a hybrid approach, which releases recognition when motion at the client-side is low. We demonstrated two sample applications for such a mobile recognition system, namely a slide recognition system for meeting rooms and a mobile tourist guide. The latter included several types of geographic information to restrict the search space. The mobile applications were complemented with two applications for the desktop or the web, namely auto-annotation and 3D reconstruction – both applications build directly on the results from Chapter 4. Finally, we introduced a method to localize and read text in natural images, with the goal to make such cues available to a retrieval system. The method follows the Viola-Jones approach for face detection, but adapted to the problem of text detection by using different feature sets. The recognition capabilities were evaluated on challenging data of natural scenes, partly taken with mobile phone cameras. It

turned out that the localization method is very robust, but OCR on the extracted text regions turned out to be more challenging than expected.

In Chapter 6 we evaluated methods which allow to scale object-level retrieval to large amounts of data in the order of up to 1 million images. We investigated which properties make nearest neighbor search for databases of local features different from "general purpose" nearest neighbor search. We then evaluated three methods (LSH, Redundant Bit Vectors, and Metric Trees) under that aspect on benchmark data. It turned out that metric trees offered the best trade-off between precision, scalability and ease of handling in this evaluation. Therefore we investigated improvements to scale retrieval to larger amounts of data while maintaining precision. This was achieved by combining several metric trees into forest, where each tree was built from a random data sample. The quality of this approach was evaluated and compared to state of the art methods on several benchmark datasets and showed competitive precision and recall values for retrieval tasks of up to one million images.

7.2 Perspectives

The following perspectives for extension of the work presented in this thesis seem worth investigating:

Itemset mining in visual data has potential for several extensions both at the detail level and in a wider context. Detailed improvements include: the refinement of the semi-local neighborhood. Here, building on rotation instead of scale could potentially be more robust, since this cue is more robust in the underlying feature detectors. The spatial tiling could be extended to a spatial pyramid, or multiresolution histogram, respectively. This would allow capturing spatial constraints at several levels. The calculation of semi-local neighborhood transactions could be made very efficient by using an integral-histogram inspired approach. In a wider context, massively parallel deployments as proposed recently in [Li *et al.*, 2008a] and tests on large datasets from the Internet could be interesting. In that context, semi-supervised recognition systems would be particularly exciting. It might be worthwile to revive the recently somewhat neglected concept of relevance feedback, but using local features instead of global ones. Simple methods such as itemset mining could then potentially be used to learn structural patterns of features for a given query on-line.

Mining objects and events from community photo collections has enormous further potential. The amount of (geotagged) photos available is growing constantly and rapidly, which allows crawling and processing of enormous amounts of data and

consequently identifying many more objects and events, also in the "long tail" of data. Especially the labeling of the identified clusters could be extended, using multimodal classification methods. Interesting directions of future research could be scene classification (indoor/outdoor/day/night/weather *etc.*) or the analysis of events at a visual and textual level, *e.g.* events such as weddings could potentially be learned based on visual cues. Auto-annotation on the object level offers great opportunities for exciting applications such as auto-tagging of holiday snaps for web- and desktop applications. These annotations could be improved by investigating the scene geometry in more detail and based on that derive refined annotations at a high level of detail, *e.g.* by labeling certain parts of objects individually. Combinations with other lines of work such as 3D reconstruction [Snavely *et al.*, 2006] offer further potential. Finally, transferring the concept to other databases on the Internet, *e.g.* images of products would be fascinating, too.

Multimodal retrieval applications from mobile devices can be refined in many ways. Combinations with augmented reality seem particularly fruitful, considering the rapidly improving capabilities of mobile devices. In our experience, a combined approach of server-side recognition and client-side tracking or augmentation seems the most promising. Inclusion of multimodal cues, such as geographic location will be crucial for real world deployments. A combination with databases as the ones mined with the work presented in Chapter 4 is a natural next step. The work on text detection showed, that even a very well researched field such as OCR has further potential for improvement and novel applications, when applying it to natural scenes instead of scanned documents.

Scalable retrieval for local features is a topic which receives a lot of attention currently. Focussing on the special characteristics introduced by building on local features instead of global ones seems promising. The ultimate indexing method would probably combine appearance of the features and their geometric arrangement to achieve both better precision and scalability. For Internet Vision applications the combinations of database mining as in Chapter 4 an scalable indexing methods is very promising. The information collected on the objects (multiple views *etc.*) could be helpful in improving visual vocabularies.

In summary, closing the circle to the introductory statements of this thesis, we investigated several directions in which computer vision methods allow for organizing and searching repositories of visual data. The combination of mined specific objects, cues for the presence of instances of object classes, and detected text will allow for creating systems, which decompose and label many common scenes captured in photos. The trends outlined in the introduction of this thesis are now tackled in the rising research field "Internet Vision", which offers both great challenges and

opportunities for further research and applications along some of the directions touched in this dissertation.

A

Amazon Example Results

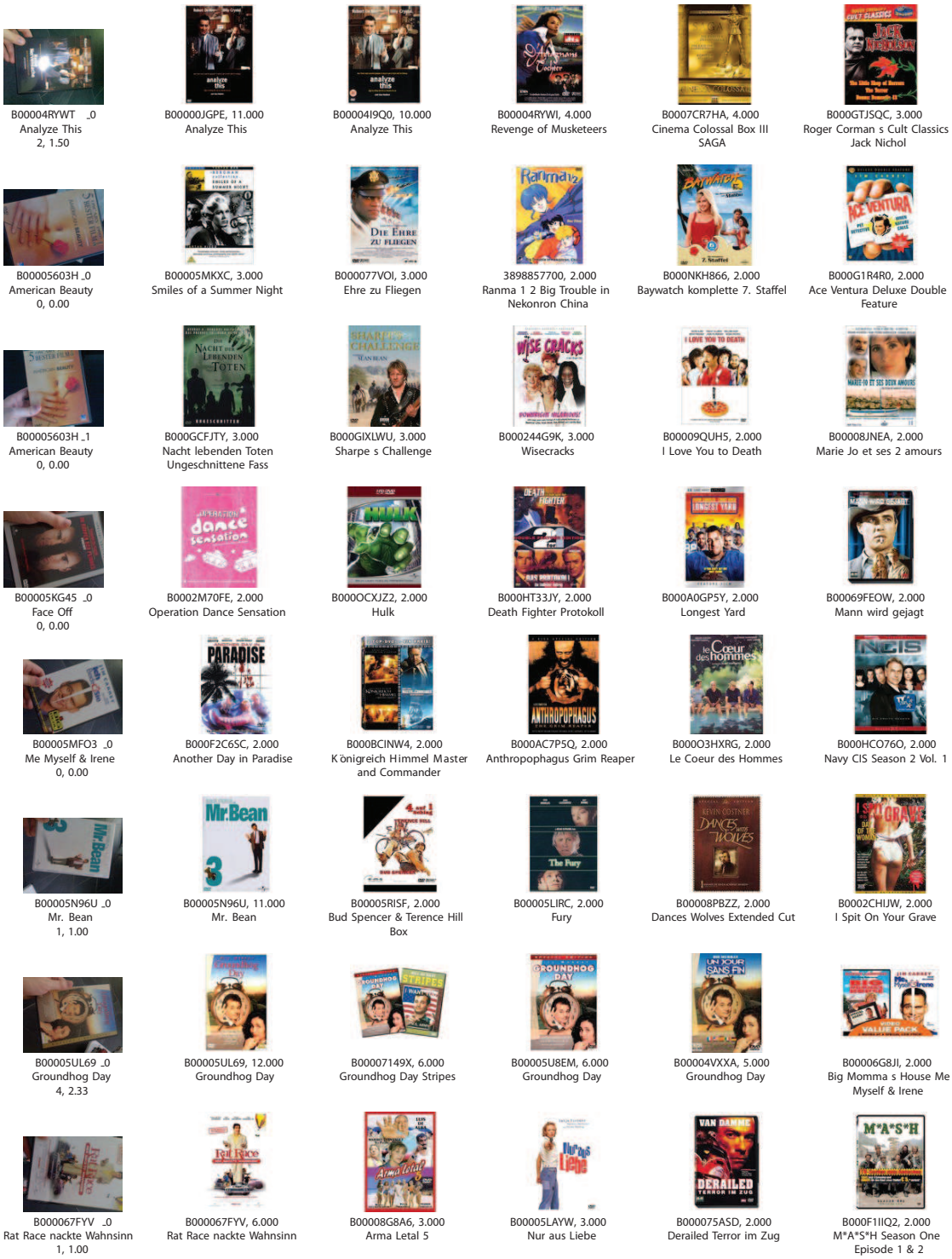


Figure A.1: *Amazon Example Results (I). The leftmost image on each line is the query image, the remaining images are results, sorted by rank from left to right.*

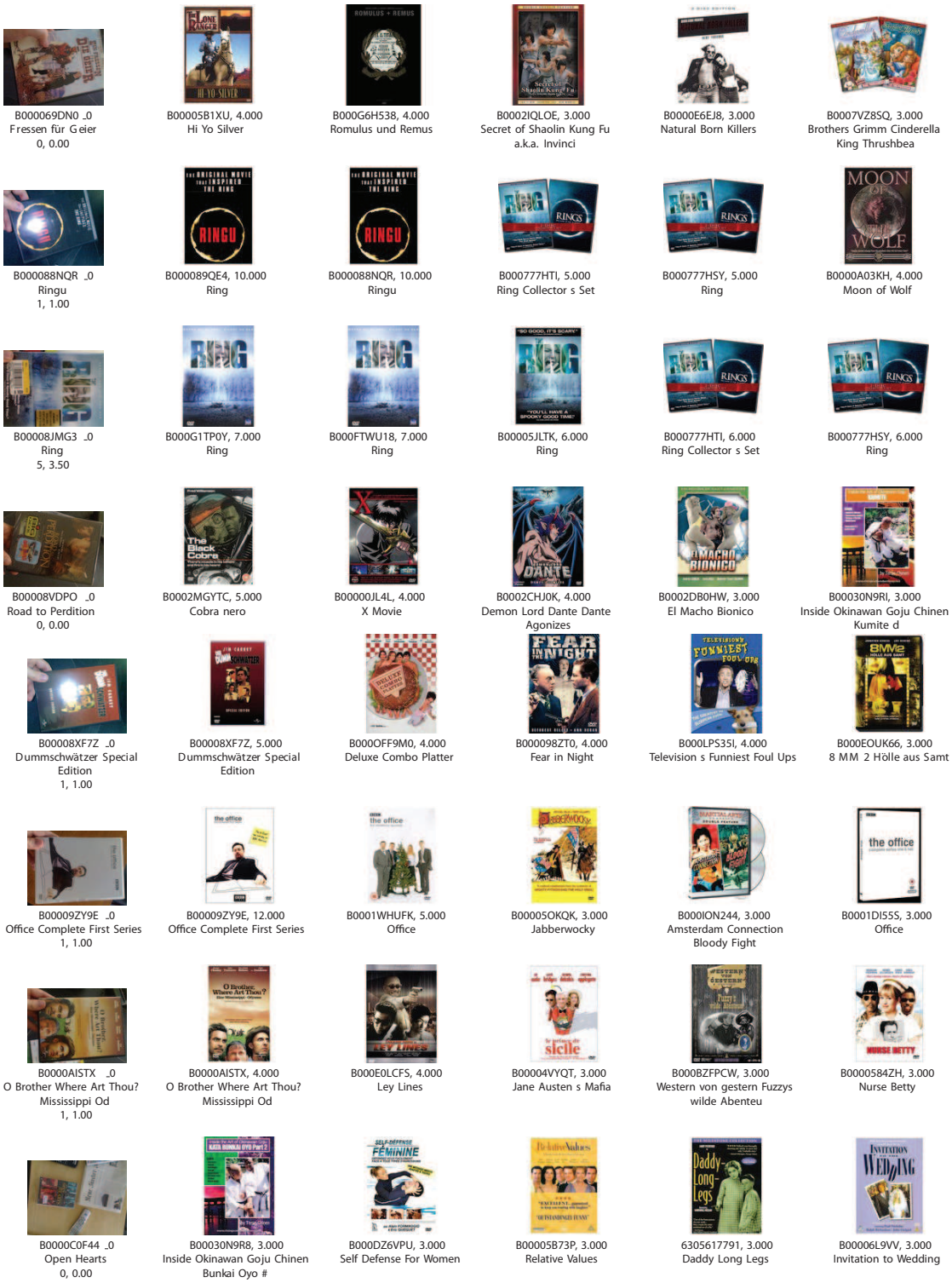


Figure A.2: *Amazon Example Results (II).The leftmost image on each line is the query image, the remaining images are results, sorted by rank from left to right.*

Bibliography

[Abowd, 1999] G. Abowd. Classroom 2000: An experiment with the instrumentation of a living educational environment. In *IBM Systems Journal*, 1999.

[Adelmann *et al.*, 2006] R. Adelmann, M. Langheinrich, and C. Floerkemeier. A toolkit for bar-code-recognition and -resolving on camera phones – jump starting the internet of things. In *Workshop Mobile and Embedded Interactive Systems (MEIS'06) at Informatik 2006*, 2006.

[Agarwal and Roth, 2002] Shivani Agarwal and Dan Roth. Learning a sparse representation for object detection. In *ECCV02*, 2002.

[Agarwal *et al.*, 2004] Shivani Agarwal, Aatif Awan, and Dan Roth. Learning to detect objects in images via a sparse, part-based representation. In *Trans. PAMI*, 2004.

[Aggarwal and Yu, 1998] C. C. Aggarwal and P. S. Yu. A new framework for itemset generation. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1998.

[Agrawal *et al.*, 1993] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD'93*, 1993.

[Amir *et al.*, 2001] A. Amir, G. Ashour, and S. Srinivasan. Toward automatic real time preparation of online video proceedings for conference talks and presentations. In *Hawaii Int. Conf. on System Sciences*, 2001.

[Antonie *et al.*, 2003] M. Antonie, O. Zaïane, and A. Coman. Associative classifiers for medical images. In *Lecture Notes in A.I. 2797, Mining Multimedia and Complex Data*, 2003.

[Aurnhammer *et al.*, 2006] M. Aurnhammer, P. Hanappe, and L. Steels. Integrating collaborative tagging and emergent semantics for image retrieval. In *Collaborative Web Tagging Workshop (WWW'06)*, 2006.

[Ballagas *et al.*, 2005] Rafael Ballagas, Michael Rohs, and Jennifer G. Sheridan. Mobile phones as pointing devices. In *PERMID '05*, 2005.

[Ballard, 1981] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[Bay *et al.*, 2006a] H. Bay, B. Fasel, and L. Van Gool. Interactive museum guide: Fast and robust recognition of museum objects. In *Proc. Intern. Workshop on Mobile Vision*, 2006.

[Bay *et al.*, 2006b] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *ECCV'06*, 2006.

[Benzécri, 1982] J.P. Benzécri. Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques. *Cahiers de l'Analyse des Données*, 7(2):209–218, 1982.

[Borenstein and Ullman, 2002] E. Borenstein and S. Ullman. Class-specific, top-down segmentation. In *ECCV02*, 2002.

[Borgelt and Berthold, 2002] Christian Borgelt and Michael R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. *ICDM*, 00:51, 2002.

[Borgelt, 2003] Christian Borgelt. Efficient implementations of apriori and eclat. In *Workshop of Frequent Item Set Mining Implementations (FIMI 2003)*, 2003.

[Borgelt, 2005] C. Borgelt. An implementation of the fp-growth algorithm. In *OSDM'05*, 2005.

[Boring *et al.*, 2007] Sebastian Boring, Manuela Altendorfer, Gregor Broll, Otmar Hilliges, and Andreas Butz. Shoot & copy: Phonecam-based information transfer from public displays onto mobile phones. In *International Conference on Mobile Technology, Applications and Systems*, 2007.

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

[Breu and Müller, 2008] M. Breu and M. Müller. A motion-detection based user interface for mobile visual search. Semester Project, 2008.

[Brin *et al.*, 1997] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD Rec.*, 26(2):255–264, 1997.

[Burl *et al.*, 1998] M. C. Burl, M. Weber, and P. Perona. A probabilistic approach to object recognition using local photometry and global geometry. In *ECC98*, 1998.

[Carletta et al. (17 authors), 2005] J. Carletta et al. (17 authors). The ami meeting corpus: A pre-announcement. In *MLMI*, 2005.

[Carson *et al.*, 1999] C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. In *Third International Conference on Visual Information Systems*, 1999.

[Chen and Yuille, 2004] Xiangrong Chen and A.L. Yuille. Detecting and reading text in natural scenes. In *CVPR04*, 2004.

[Chum and Zisserman, 2007] O. Chum and A. Zisserman. An exemplar model for learning object classes. In *CVPR07*, 2007.

[Ciaccia *et al.*, 1997] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB '97*, 1997.

[Cooley *et al.*, 1993] R. Cooley, J. Srivastava, and B. Mobasher. Web mining: Information and pattern discovery on the world wide web. In *ICTAI*, 1993.

[Cox *et al.*, 2000] Ingemar J. Cox, Matthew L. Miller, Thomas P. Minka, Thomas Papathomas, and Peter N. Yianilos. The bayesian image retrieval system, pichunter: Theory, implementation and psychophysical experiments. *IEEE Transactions on Image Processing (to appear)*, 9(1):20–37, January 2000.

[Croft and Harper, 1997] W. B. Croft and D. J. Harper. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation*, 35, 1997.

[Dalal and Triggs, 2005] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR'05*, 2005.

[Dance *et al.*, 2004] C. Dance, J. Willamowski, L. Fan, C. Bray, and G. Csurka. Visual categorization with bags of keypoints. In *ECCV SLCV'04*, 2004.

[Datar *et al.*, 2004] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG '04*, 2004.

[Davis, 2001] J.W. Davis. Hierarchical motion history images for recognizing human motion. In *Proceedings of IEEE Workshop on Detection and Recognition of Events in Video, 2001*, pages 39–46, 2001.

[de Rham, 1980] C. de Rham. La classification hiérarchique ascendante selon la méthode des voisins r éciproques. *Cahiers de l'Analyse des Données*, 5(2):135–144, 1980.

[Dufour *et al.*, 2002] R.M. Dufour, E.L. Miller, and N.P. Galatsanos. Template matching based object recognition with unknown geometric parameters. *Image Processing, IEEE Transactions on*, 11(12):1385–1396, Dec 2002.

[Edwards, 2004] L. Edwards. *Developing Series 60 Applications*. Addison Wesley, 2004.

[Elkan, 2003] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML'03*, pages 147–253, 2003.

[Farnstrom *et al.*, 2000] Fredrik Farnstrom, James Lewis, , and Charles Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2(1):51—57, 2000.

[Fei-Fei *et al.*, 2003] L. Fei-Fei, R. Fergus, and P. Perona. A bayesian approach to unsupervised one-shot learning of object categories. In *ECCV03*, 2003.

[Fei-Fei *et al.*, 2004] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an approach tested on 101 object categories. In *CVPR WGMBV*, 2004.

[Feltzenswalb and Hutenlocher, 2005] P. Feltzenswalb and D. Hutenlocher. Pictorial structures for object recognition. *IJCV*, 61(1), 2005.

[Fergus *et al.*, 2003] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *CVPR'03*, 2003.

[Fergus *et al.*, 2005] R. Fergus, P. Perona, and A. Zisserman. A sparse object category model for efficient learning and exhaustive recognition. In *CVPR'05*, 2005.

[Ferrari *et al.*, 2006] V. Ferrari, T. Tuytelaars, and L. Van Gool. Object detection by contour segment networks. In *ECCV'06*, 2006.

[Fischler and Bolles., 1981] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Comm. of the ACM*, 1981.

[Flickner *et al.*, 1995] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petković, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23–32, September 1995.

[Föckler *et al.*, 2005] Paul Föckler, Thomas Zeidler, Benjamin Brombach, Erich Bruns, and Oliver Bimber. Phoneguide: museum guidance supported by on-device object recognition on mobile phones. In *MUM '05: Proceedings of the 4th international conference on Mobile and ubiquitous multimedia*, 2005.

[Freund and Schapire, 1997] Y. Freund and R Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[Fritz *et al.*, 2005] M. Fritz, B. Leibe, and B. Caputoand B. Schiele. Integrating representative and discriminant models for object category detection. In *ICCV'05*, 2005.

[Fuhrmann and Harbaum, 2003] T. Fuhrmann and T. Harbaum. Using bluetooth for informationally enhanced environments. In *Proceedings of the IADIS International Conference e-Society 2003*, 2003.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[Goesele *et al.*, 2007] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. Seitz. Multi-view stereo for community photo collections. In *ICCV'07*, 2007.

[Goldstein *et al.*, 2005] Jonathan Goldstein, John C. Platt, and Christopher J.C. Burges. Redundant bit vectors for quickly searching high-dimensional regions. In *Det. and Stat. Methods in Machine Learning*, 2005.

[Griffin *et al.*, 2007] G. Griffin, A.D. Holub, and P. Perona. The caltech 256. Caltech Technical Report, 2007.

[Grimson and Lozano-Pérez, 1987] W. E. L. Grimson and T. Lozano-Pérez. Localizing overlapping parts by searching the interpretation tree. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(4):469–482, 1987.

[Gugl, 2007] S. Gugl. Object class recognition by frequent graph mining. Master's thesis, Computer Vision Institute, ETH Zurich, 2007.

[Han *et al.*, 2000] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000.

[Hand *et al.*, 2001] D.J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, 2001.

[Hand, 2001] D.J. Hand. *Principles of Data Mining*. MIT Press, 2001.

[Harris and Stephens, 1988] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, 1988.

[Hartley and Zisserman, 2004] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge Univ. Press, 2004.

[Hays and Efros, 2008] J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *CVPR08*, 2008.

[Holt and Chun, 1999] John D. Holt and Soon Myoung Chun. Efficient mining of association rules in text databases. In *ACM CIKM*, 1999.

[Inokuchi *et al.*, 2003] A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50:321—354, 2003.

[Jaffe *et al.*, 2006] A. Jaffe, M. Naaman, T. Tassa, and M. Davis. Generating summaries and visualization for large collections of geo-referenced photographs. In *MIR'06*, 2006.

[Jain and Dubes, 1988] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[Jecker and Knecht, 2008] Raphael Jecker and Benjamin Knecht. Real-time server-side object recognition for mobile devices. Semester Project, 2008.

[Jung *et al.*, 2004] Keechul Jung, Kwang In Kim, and Anil K. Jain. Text information extraction in images and video: a survey. *Pattern Recognition*, 37(5):977–997, 2004.

[Kamvar and Baluja, 2006] M. Kamvar and S. Baluja. A large scale study of wireless search behavior: Google mobile search. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, 2006.

[Kaufman and Rousseeuw, 1990] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.

[Kuramochi and Karypis, 2001] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM'01: 1st IEEE Conf. Data Mining*, pages 313–320, 2001.

[Kuramochi and Karypis, 2004] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1038–1051, 2004.

[Lazebnik *et al.*, 2004] S. Lazebnik, C. Schmid, and J. Ponce. Semi-local affine parts for object recognition. In *BMVC'04*, 2004.

[Lazebnik *et al.*, 2006] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR06*, 2006.

[Leibe and Schiele, 2003] B. Leibe and B. Schiele. Interleaved object categorization and segmentation. In *BMVC'03*, 2003.

[Leibe *et al.*, 2005] Bastian Leibe, Edgar Seemann, and Bernt Schiele. Pedestrian detection in crowded scenes. In *CVPR'05*, 2005.

[Leibe *et al.*, 2008] Bastian Leibe, Ales Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *IJCV Special Issue on Learning for Vision and Vision for Learning*, 2008.

[Leung and Malik, 2001] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[Levenshtein, 1966] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–?710, 1966.

[Lew *et al.*, 2006] S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. In *ACM Trans. Multimedia Comput. Commun. Appl.*, 2006.

[Lewis *et al.*, 2007] A. Lewis, M. Purvis, J. Sambells, and C. Turner. *Beginning Google Maps Applications with Rails and Ajax*. Apress, 2007.

[Li *et al.*, 2008a] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: Parallel fp-growth for query recommendation. In *ACM Recommendation Systems 08*, 2008.

[Li *et al.*, 2008b] X. Li, C. Wu, C. Zach, S. Lazebnik, and J.-M. Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *ECCV08*, 2008.

[Liu *et al.*, 2004] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *NIPS'04*, 2004.

[Liu *et al.*, 2007] T. Liu, C. Rosenberg, and H. A. Rowley. Clustering billions of images with large scale nearest neighbor search. In *WACV '07*, 2007.

[Liu, 2006] Ting Liu. Fast nonparametric machine learning algorithms for high-dimensional massive data and applications. In *PhD Thesis*, 2006.

[Lowe, 1991] David G. Lowe. Fitting parameterized three-dimensional models to images. *PAMI*, 13(5):441–450, 1991.

[Lowe, 1999] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV99*, 1999.

[Lowe, 2004] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004.

[Lu and Tan, 2007] S. Lu and C. L. Tan. Binarization of badly illuminated document images through shading estimation and compensation. In *Ninth International Conference on Document Analysis and Recognition, 2007. ICDAR 2007.*, 2007.

[Lucas, 2005] S. M. Lucas. Text locating competition results. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, 2005.

[Ma and Manjunath, 1999] Wei-Ying Ma and B. S. Manjunath. Netra: A toolbox for navigating large image databases. *Multimedia Systems*, 7(3):184–198, 1999.

[MacQueen, 1967] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In University of California Press, editor, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley*, volume 1, pages 281–297, 1967.

[Magagna, 2008] Fabio Magagna. Unsupervised 3d reconstruction from images mined in community photo collections. Master's thesis, ETH Zurich, Computer Vision Lab, 2008.

[Matas *et al.*, 2002] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. In *BMVC'02*, 2002.

[Mathes,] Adam Mathes. Folksonomies-cooperative classification and communication through shared metadata. Website (visited 15.6.2008).

[Metwally *et al.*, 2005] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Using association rules for fraud detection in web advertising networks. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, 2005.

[Mikolajczyk and Schmid, 2004a] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *IJCV*, 60(1), 2004.

[Mikolajczyk and Schmid, 2004b] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *IJCV*, 60:63–86, 1 2004.

[Mikolajczyk and Schmid, 2005] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *PAMI*, 27(10):1615–1630, 2005.

[Mikolajczyk *et al.*, 2005] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *IJCV*, 65:43–72, 2005.

[Minogue and Gondry, 2002] K. Minogue and M. Gondry. Come into my world, 2002.

[Minogue and Shadforth, 2001] K. Minogue and D. Shadforth. Can't get you out of my head, 2001.

[Moore, 2000] Andrew W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Conference on Uncertainty in Artificial Intelligence*, 2000.

[Moosmann *et al.*, 2006] F. Moosmann, B. Triggs, and F. Jurie. Randomized clustering forests for building fast and discriminative visual vocabularies. In *NIPS'06*, 2006.

[Murase and Nayar, 1995] Hiroshi Murase and Shree K. Nayar. Visual learning and recognition of 3-d objects from appearance. *IJCV*, 14(1):5–24, 1995.

[Niblack, 1999] Wayne Niblack. Slidefinder: A tool for browsing presentation graphics using content-based retrieval. In *CBAIVL '99*, 1999.

[Nistér and Stewénius, 2006] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. In *CVPR'06*, 2006.

[Nowozin *et al.*, 2007] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In *CVPR07*, pages 1–8, June 2007.

[Oliva and Torralba, 2001] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Comput. Vision*, 42(3):145–175, 2001.

[Omiecinski, 2003] Edward R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):57–69, 2003.

[Opelt *et al.*, 2003] A. Opelt, M. Fussenegger, A. Pinz, and P. Auer. Generic object recognition with boosting. In *Trans. PAMI*, 2003.

[Opelt *et al.*, 2006] A. Opelt, A. Pinz, and A. Zisserman. Incremental learning of object detectors using a visual alphabet. In *CVPR'06*, 2006.

[Ordonez and Omiecinski, 1999] Carlos Ordonez and Edward Omiecinski. Discovering association rules based on image content. In *ADL '99: Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries*, 1999.

[Osian and Van Gool, 2004] M. Osian and L. Van Gool. Video shot characterization. *Machine Vision Applications*, 15:172–177, 3 2004.

[Ozuysal *et al.*, 2007] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *CVPR'07*, 2007.

[Paletta *et al.*, 2006] L. Paletta, G. Fritz, C. Seifert, P. Luley, and A. Almer. A mobile vision service for multimedia tourist applications in urban environments. In *IEEE Intel. Transp. Syst. Conf.*, 2006.

[Pelleg and Moore, 1999] D. Pelleg and A. Moore. Accelerating exact k -means algorithms with geometric reasoning. In *Knowledge Discovery and Data Mining KDD'99*, 1999.

[Pelleg and Moore, 2000] Dan Pelleg and Andrew Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML00*, 2000.

[Philbin *et al.*, 2007] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR'07*, 2007.

[Piatetsky-Shapiro, 1991] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

[Quack *et al.*, 2004] Till Quack, Ullrich Mönich, Lars Thiele, and B. S. Manjunath. Cortina: a system for large-scale, content-based web image retrieval. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 508–511. ACM, 2004.

[Quack *et al.*, 2008] T. Quack, H. Bay, and L. Van Gool. Object recognition for the internet of things. In *Internet of Things 2008*, 2008.

[Quinlan, 1986] J.R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1:81–106, 1986.

[Renold, 2008] M. Renold. Detecting and reading text in natural scenes. Master's thesis, Computer Vision Lab, ETH Zurich, May 2008.

[Rohs and Gfeller, 2004] M. Rohs and B. Gfeller. Using camera-equipped mobile phones for interacting with real-world objects. In *Advances in Pervasive Computing, Austrian Computer Society (OCG)*, 2004.

[Rui and Huang, 1999] Y. Rui and T. S. Huang. A novel relevance feedback technique in image retrieval. In *Proceedings of 7th ACM International Conference on Multimedia (MM)*, 1999.

[Salton and McGill, 1986] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1986.

[Sandvig *et al.*, 2007] J. J. Sandvig, Bamshad Mobasher, and Robin Burke. Robustness of collaborative recommendation based on association rule mining. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, 2007.

[Scheiner and Schwarz, 2007] D. Scheiner and R. Schwarz. High performance object recognition. Master's thesis, ETH Zurich, Computer Vision Lab, 2007.

[Schmid and Mohr, 1997] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *PAMI*, 19(5):530–535, 1997.

[Shotton *et al.*, 2006] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV'06*, 2006.

[Silverstein *et al.*, 1998] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Min. Knowl. Discov.*, 2(1):39–68, 1998.

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

[Singhal *et al.*, 1996] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *SIGIR '96*, 1996.

[Sivic and Zisserman, 2003] J. Sivic and A. Zisserman. Video google: a text retrieval approach to object matching in videos. In *ICCV'03*, 2003.

[Sivic and Zisserman, 2004] Josef Sivic and Andrew Zisserman. Video data mining using configurations of viewpoint invariant regions. In *CVPR'04*, 2004.

[Sivic *et al.*, 2005] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman . Discovering object categories in image collections. In *CVPR05*, 2005.

[Smith, 2004] G. Smith. Folksonomy: social classification. Website (visited 15.6.2008), 8 2004.

[Snavely *et al.*, 2006] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. on Graphics*, 25(3), 2006.

[Spirito *et al.*, 2001] M.A. Spirito, S. Pöykkö, and O. Knuuttila. Experimental performance of methods to estimate the location of legacy handsets in gsm. In *IEEE Veh. Technol. Conf., 2001*, 2001.

[Swain and Ballard, 1991] Michael J. Swain and Dana H. Ballard. Color indexing. *IJCV*, 7(1):11–32, 1991.

[Takacs *et al.*, 2008] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W-C. Chen;, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In

ACM International Conference on Multimedia Information Retrieval (MIR'08), 2008.

[Tan *et al.*, 2002] P. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[Tesic *et al.*, 2003] Jelena Tesic, Shawn Newsam, and Bangalore S. Manjunath. Mining image datasets using perceptual association rules. In *SIAM Sixth Workshop on Mining Scientific and Engineering Datasets*, 2003.

[Torralba *et al.*, 2008] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR08*, June 2008.

[Tuytelaars and Mikolajczyk, 2008] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.

[Uhlmann, 1991] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. In *Inf. Proc. Lett*, 1991.

[Ulrich, 2006] Tamara Ulrich. Object recognition on a mobile phone: Part ii. Semester Project, 2006.

[Valiant, 1984] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27, 1984.

[Various, 2005] Various. The pascal object recognition database collection (2005), 2005. www.pascal-network.org/challenges/VOC.

[Vergauwen and Van Gool, 2006] M. Vergauwen and L. Van Gool. Web-based 3d reconstruction service. *MVA*, 17(6):411–426, 2006.

[Vinciarelli and Odobez, 2006] A. Vinciarelli and J. Odobez. Application of information retrieval technologies to presentation slides. *IEEE Transactions on Multimedia*, 8(5):981–995, 2006.

[Viola and Jones, 2001a] P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In *NIPS01*, 2001.

[Viola and Jones, 2001b] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR01*, 2001.

[Wagner *et al.*, 2008] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Pose tracking from natural features on mobile phones. In *ISMAR'08*, 2008.

[Wal, 2005] Thomas Vander Wal. Folksonomy definition and wikipedia. Website, 11 2005.

[Want, 2004] R. Want. Rfid - a key to automating everything. In *Scientific American*, 2004.

[Washio and Motoda, 2003] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, 2003.

[Webb, 2002] A. Webb. *Statistical Pattern Recognition*. Wiley, second edition, 2002.

[Weber *et al.*, 2000a] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *CVPR00*, 2000.

[Weber *et al.*, 2000b] Markus Weber, Max Welling, and Pietro Perona. Unsupervised learning of models for recognition. In *ECCV00*, 2000.

[Wiskott *et al.*, 1997] Laurenz Wiskott, Jean-Marc Fellous, Norbert Kruger, and Christoph von der Malsburg. Face recognition by elastic bunch graph matching. *PAMI*, 19(7):775–779, 1997.

[Wolfson and Rigoutsos, 1997] Haim J. Wolfson and Isidore Rigoutsos. Geometric hashing: An overview. *IEEE Comput. Sci. Eng.*, 4(4):10–21, 1997.

[Wu, 2005] Ching-Tung Wu. Embedded-text detection and its application to anti-spam filtering. Master's thesis, University of California, Santa Barbara, 2005.

[Yan and Han, 2002] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, 2002.

[Yan and Han, 2003] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003.

[Yang, 2006] Guizhen Yang. Computational aspects of mining maximal frequent patterns. *Theor. Comput. Sci.*, 362(1):63–85, 2006.

[Zaiane *et al.*, 1998] Osmar R. Zaiane, Jiawei Han, Ze-Nian Li, and Jean Hou. Mining multimedia data. In *CASCON'98*, 1998.

[Zaki, 2000] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[Zobel and Moffat, 2006] Justin Zobel and Alistair Moffat. Inverted files for text search engines. In *ACM Comput. Surv.*, 2006.

Curriculum Vitae

Personal Data

Name	Till Quack
Date of Birth	15.09.1978
Place of Birth	Göttingen, Germany
Citizenship	German

Education

1984 – 1991	Primary School. Ebmatingen and Pfaffhausen, Switzerland
1991 – 1998	High-School. Realgymnasium Rämibühl, Zürich, Switzerland
1998 – 2004	Studies of Information Technology and Electrical Engineering at ETH Zurich, Switzerland. Graduation: MSc. ETH and Dipl. Ing. ETH in Information Technology and Electrical Engineering
Fall 2004	MSc. Project at University of California at Santa Barbara. Vision Research Lab
2004 – 2008	Doctoral Student at ETH Zurich, Computer Vision Laboratory, Department of Information Technology and Electrical Engineering

Occupations

1998	Marent AG, Technical Author
1998 – 2006	Quack Internet Solutions, founder
2004 – 2008	Research Assistant at ETH Zurich, Computer Vision Laboratory, Department of Information Technology and Electrical Engineering
2006 –	kooaba AG, co-founder & CTO