Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehrstuhl für Informatik VIII (Computergraphik, Computer Vision und Multimedia)
Mobile Multimedia Processing
Prof. Dr. Bastian Leibe

Diplomarbeit

Towards Large-Scale Categorization Using Min-Hash

Jan Hendrik Hosang
Matrikelnummer: 267687

August 2011

Erstgutachter: **Prof. Dr. Bastian Leibe**
Zweitgutachter: **Prof. Dr. Bernt Schiele**

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

Aachen, August 18, 2011

(Jan Hendrik Hosang)

Abstract

The visual knowledge about the world's objects, which is publicly available in internet image collections, is growing at ever faster rates. Trying to keep up with this increasing data volume, traditional holistic categorization approaches are struggling to scale up to millions of images. At the same time, the number of categories increases, making decisions and models more complex. Fast image retrieval methods allow fast categorization using both machine learning methods that operate on the local neighborhood and local ad-hoc approximations of holistic approaches. In particular, Min-Hash promises the retrieval of highly similar images in constant time, i.e. independent of the number of indexed images. In this work, we investigate the potential of Min-Hash for near neighbor search in a large-scale categorization setting. We evaluate Min-Hash and several extensions on the ImageNet Large Scale Visual Recognition Challenge 2010 using nearest neighbor categorization. Extensions include tf-idf weighting, permutation grouping, and Geometric Min-Hash as well as a novel generalization of Geometric Min-Hash for small vocabularies.

Acknowledgment

Wow, this project grew much bigger than I ever imagined. Many thanks go to Prof. Dr. Bastian Leibe for the cunning plan and for setting up the collaboration with the Max Planck Institute in Saarbrücken. I would also like to thank Prof. Dr. Bernt Schiele for accepting the project and offering the computing resources of his institute. I thank both for the interesting opportunity and for supervising my work.

I would like to express my gratitude to Tobias Weyand, who has been my supervisor since November 2009 when I started as a student assistant. Thank you for your patient supervision and your motivating and frequent discussions.

Further, I would like to thank Sandra Ebert and Mario Fritz for fortnightly telephone conferences, which I secretly called the "think tank". You offered long critical discussions and sparked new ideas and different perspectives.

Last and certainly not least, I would like to thank Bastian Leibe again for frequent discussions and assessments of new methods as well as coming up with totally new ideas.

Contents

1 Introduction **1**

2 Related Work **3**

3 Local Features for Retrieval and Categorization **7**
3.1 Types of Descriptors 7
3.2 Interest Regions 9
3.2.1 The Hessian Detector 9
3.2.2 The Hessian-Laplace Detector 10
3.2.3 Affine Covariant Region Detection 11
3.3 Interest Region Descriptors 13
3.3.1 The SIFT Descriptor 13
3.4 Bags of Visual Words 14
3.4.1 Vocabulary Construction 14
3.4.2 Distance Measures 17

4 Locality Sensitive Hashing **19**
4.1 Min-Hash 20
4.1.1 Collision Probability Amplification 22
4.1.2 Word Weighting 23
4.1.3 Term Frequency Weighting 26
4.1.4 Permutation Grouping 27
4.1.5 Threshold Adaption 30
4.2 Geometric Min-Hash 31
4.2.1 Analysis of the Collision Probability 33
4.2.2 Review of Min-Hash Extensions 34
4.2.3 Multi-Region Geometric Min-Hash 35
4.2.4 Deleting Used Words 37
4.3 Re-ranking 37

5 ImageNet Database **41**
5.1 ImageNet 41
5.1.1 Construction 42

5.1.2 Statistics . . . 43
5.2 Large Scale Visual Recognition Challenge 2010 . . . 43
5.2.1 Examples . . . 44
5.2.2 Error Measures . . . 49
5.2.3 The *tiny* Set . . . 51

6 Pipeline Design **53**
6.1 Pipeline Overview . . . 53
6.1.1 Feature Extraction . . . 54
6.1.2 Hash Table Construction . . . 55
6.1.3 Hash Table Queries . . . 57
6.1.4 Ranking and Categorization . . . 58
6.2 Implementation Details . . . 58
6.2.1 Infrastructure Analysis . . . 59
6.2.2 Programming Language and File Formats . . . 60

7 Experiments and Evaluation **63**
7.1 Baseline Setups . . . 63
7.1.1 Guessing . . . 64
7.1.2 Support Vector Machines . . . 65
7.2 Reference Setup: Exact k-Nearest Neighbor . . . 66
7.3 Effect of Hashing Parameters . . . 71
7.3.1 Collision Rate . . . 71
7.3.2 Approximation Quality . . . 73
7.3.3 Frequency Analysis of Features . . . 76
7.4 Min Hash . . . 78
7.4.1 tf-idf Min Hash . . . 80
7.4.2 Permutation Grouping . . . 81
7.4.3 Threshold Adaption . . . 83
7.5 Geometric Min-Hash . . . 84
7.6 Validation Results . . . 86
7.7 Conclusion . . . 89

8 Conclusion and Future Work **91**

Glossary **93**

List of Figures **95**

List of Tables **97**

Bibliography **99**

Chapter 1

Introduction

Computer vision is the art of teaching computers to do what is so common to humans that they do it entirely unconsciously: understanding what they see. "Understanding" is a deliberately vague term, because there is a broad range of actions that is associated with seeing.

The human vision system is capable of very accurate tasks. We are able to notice tiny changes, we can distinguish very similar faces and recognize familiar faces in a massive crowd without consciously searching for someone. We do all of that in the blink of an eye. Humans are also able to perform more abstract tasks just as quickly. We know the function or the category of an object at several levels of generality. Even if we never saw a certain flower before, we are able to intuitively recognize it as such.

These different tasks come so naturally to humans that it is surprising how difficult it is to solve these tasks with a computer. "The problem of computer vision" was a summer project assigned to an undergraduate student by Seymour Papert in 1966. Researchers work on related problems since decades and although significant progress has been made, many techniques are still being improved and many problems remain unsolved.

The coarse high-level goals in computer vision that are related to "determining what is in an image" are distinguished into the following categories:

Recognition is the task of distinguishing different instances of a certain category. For example, face recognition deals with recognizing individual persons. This means that we want to fade down the peculiarities of the category and concentrate on the intra-category variation.

Detection is the problem of finding instances of particular categories, e.g. pedestrians. In this example, the difference to recognition is that we do not care about who the person is. Instead, we want to determine whether a person is in the image and if so, where. So, we need to neglect the details of specific instances and concentrate on differences between foreground and background.

Categorization involves generic methods that can learn to distinguish between arbi-

trary categories. Examples of every category are presented to the algorithm and it has to learn how to discriminate between the categories. In contrast to recognition and detection, it is not feasible to employ specialized methods for every category. The goal is to develop methods that are powerful enough to discriminate between any categories.

During the recent years, many achievements are due to the accessibility of large amounts of images via the internet and decreasing prices of image acquisition hardware. For instance, the popular image hoster flickr contains more than 5 billion photos as of September 2010, increasing at a rate of about 3,000 images per minute. Since 2000, 4.2 billion camera phones have been sold, fueling the increasing rate at which images are uploaded to the internet.

A popular application of categorization is automatic image tagging for semantic indexing. For example, Google or flickr image search provide a text-based search through large image databases, but they only take account of manually assigned tags or the text of the website around images. A fully automated system could provide semantics autonomously.

Goal

In this work, we aim to improve the search through large amounts of data for categorization. Classification methods usually benefit from more training data, yet the amount of training data has a substantial impact on the run-time of the training procedure. Using an extensive database, we hope to be able to retrieve relevant images without searching through the entire database.

Search for recognition differs from search for categorization. The former aims at finding images of the same object, while the latter cannot rely on the exact query object to be contained in the database. Thus, the search has to be designed to be less specific and retrieve similar images quickly without returning too many unrelated images. Specifically, we employ the Locality Sensitive Hashing techniques Min-Hash [CPIZ07] and Geometric Min-Hash [CPM07], which have a constant query-time that is independent of the size of the database.

Our goal is to investigate how useful results of hashing are for categorization and if the performance can be improved by leaving the traditional parameter ranges. We want to examine weaknesses of the methods and propose extensions to compensate for their shortcomings.

In future work, these methods should provide the possibility to be employed for near neighbor search for local methods or for local approximations of global methods. In this case, local methods mean algorithms that operate on the neighborhood of a query, such as k-nearest neighbor, and global methods mean algorithms that require the entire dataset for training, for instance Support Vector Machines. We do not aim at a fully tuned system in terms of run-time or error rate, but attempt to explore new methods that have potential and are worth being investigated.

Chapter 2

Related Work

In this chapter, we give an overview of the development of vision algorithms towards large scale image categorization. We point out how problems were solved and which new challenges were discovered.

Global Representations

One of the first approaches that models object appearance was the *Eigenface* method [TP91], an algorithm for face recognition. The approach employs raw pixel values as image representation and applies Principal Component Analysis on a portrait collection to compute the basis of a "face space". The face space has a much lower dimensionality than the original images and minimizes the reconstruction error of the training images. An extension called *Fisherfaces* [BHK96] optimizes the class separability instead of the reconstruction error, using Fisher's Linear Discriminant. Therefore, Fisherfaces are discriminative and can be trained to detect e.g. whether a person wears glasses. The downside of both methods is that all faces in the images need to be perfectly aligned. If a face is tilted or shifted, many pixel values change, the correspondences of particular parts of a face in the global representation are lost. From a general categorization perspective, this method is only capable of distinguishing objects with a low variance in appearance, such as faces.

Histograms provide translation and rotation invariance, which was leveraged for classification in [SB91]. The approach uses color histograms as global image representations and histogram intersection as a distance measure. To localize detected objects, *histogram backprojection* was introduced. Color histograms are partly robust to perspective changes, lighting conditions and partial occlusions, yet they are distracted by clutter and require objects that have distinctive color schemes.

Another interesting application of histograms for global representations was employed for texture classification [VZ05]. The method convolves every training image with a filterbank, clusters the responses with with k-means and inserts the resulting cluster centers into a vocabulary of textons. An image representation can then be

obtained by replacing filter responses of an image by the most similar textons and entering them into a histogram. Classification is done using k-nearest neighbor with a χ^2 kernel.

Eigenfaces and Fisherfaces are model-based approaches which need updating in case of new training images. In contrast to that, histograms are model-free and new training data can simply be entered into histograms without the need of training a new model. While global image descriptors offer a terse representation of the global structure of an image, they lack translation invariance and robustness to strong viewpoint changes, clutter, occlusions, or strongly articulated objects.

Local Representations

Schiele and Crowley extended histograms as image representations to multiple dimensions [SC00]. The multidimensional histograms estimate joint distributions of filterbank responses. The remarkable contribution of the work is that it overcomes the shortcomings of global descriptors by modeling the posterior probability of an image based on local features. Small patches are represented as filter responses and their likelihood is modeled in multidimensional histograms. Interestingly, an early form of inverted file retrieval is used, called *local appearance hashing*.

A basic paradigm shift in the computer vision community is due to Lowe [Low04]. His work introduced a complete pipeline for specific object recognition, which still constitutes the state of the art in large parts because it incorporates a large amount invariance. The pipeline employed interest points and feature descriptors for a local image representation. Lowe chose an approximate nearest neighbor search using k-d trees for feature matching, and also proposed a criterion for detecting false feature matches. The matches are used to estimate an affine transformation between query and retrieved images. This approach is model-free, but it has to perform a costly matching between a query and every training image. A matching involves an approximate nearest neighbor search for every local feature in the query image.

To overcome those scalability issues, Sivic and Zisserman developed *Video Google* [SZ03] analogously to the Google text retrieval system. They quantized features and treated the resulting feature indices as "visual" words. On this basis, it is possible to employ *tf-idf* weighted retrieval. While this method improved the matching speed, vocabulary construction was still cumbersome and small vocabularies introduce quantization effects.

Vocabulary Trees [NS06] were developed for fast feature matching and larger vocabularies than before. The trees are constructed using *hierarchical k-means*, allowing fast feature quantization. Philbin et al. [PCI$^+$07] proposed a method for working on larger image corpora and for increasing the vocabulary size to one million to avoid quantization effects. Large vocabularies are obtained by randomized k-d trees and k-means. Candidate search is sped up using inverted file indexing. The search results are refined by a spatial verification using RANSAC. The results of the preceding methods

are very specific and model-free, but they solve detection of specific objects. The variance of appearance of deformable or articulated objects is too high for this approach.

The *Implicit Shape Model* [LLS08] is a probabilistic extension of the *Generalized Hough Transform*. The method jointly detects and localizes objects and provides a soft segmentation. It can cope well with clutter, intra-class variability and articulated objects, such as cows. Yet it requires clean training images from a controlled environment and has limited scalability. It also employs a visual vocabulary, but in contrast to the previous methods it is model-based.

Spatial Pyramids [LSP06] are an extension for the bag of visual words representation for applying them to categorization tasks. Images are repeatedly partitioned into rectangular regions and densely sampled local features are pooled into individual bags for each of the resulting regions. The resulting bags are simply concatenated to form one large representation in which sections capture statistics about regions of different shapes and sizes. This method reintroduces a problem of global representations: translation and rotation invariance is lost due to the subdivision of images. Moreover, this approach is model-based, specifically, it requires the classifier to be aware of the structure of the final representation. Typically, a Support Vector Machine with a Pyramid Match Kernel is employed.

Recent Work

Traditional categorization databases contain at most 300 categories and a few thousand images. For example, the Caltech256 database [GHP07] contains 256 categories and 30,000 images. Large-scale categorization experiments became possible when ImageNet offered over a million images annotated with thousands of categories. The database was used for the ImageNet Large Scale Visual Recognition Challenge 2010 (ILSVRC2010), which sparked research on this scale. Most notably, Lin et al. [LLZ$^+$11] constructed a pipeline that achieved the best performance during the challenge. Using HOG [DT05] and LBP [OPM02] features they captured both shape and texture of image regions. The resulting descriptors are coded using Local Coordinate Coding [YZG09] and Super-Vector Coding [ZYZH10]. Codes are pooled in a Spatial Pyramid, resulting in feature vectors of a dimensionality between 81,000 and 262,000. The final features are classified using a Support Vector Machine which was trained with a parallel *averaging stochastic gradient descent* algorithm [LLZ$^+$11].

The method yields very good results, yet it has to be noted that the solution by Lin et al. is based on standard approaches and massive computing power. The modification to Support Vector Machine training is a consequence of a full-blown feature representation. In contrast to this approach, we want to explore possibilities of more lightweight methods using interest regions.

Hashing

Another set of approaches from the field of image retrieval and near duplicate detection are hashing techniques. Locality Sensitive Hashing is a probabilistic method that hashes an image representation to a single hash value. The important property of the hash function is that similar images are more likely to be hashed to the same value than dissimilar images. Thus, near neighbors are likely to be among the hash collisions. Instead of searching through the entire database, we can alternatively search through the collisions, which we can determine by a lookup in a hash table.

Various Locality Sensitive Hashing methods have been developed to approximate different distance measures. For instance, *Pyramid Match Hashing* [Gra07] is a family of hash functions that approximates spatial pyramid matching. Other methods include χ^2 distance approximation or even a generic approach for arbitrary similarity functions [KG09].

Min-Hash has been adopted from text retrieval and is employed for exploring large datasets [CJ10]. It has proven useful for retrieving near duplicates, cluster seeds, and for discovering small objects [PCM09]. For a large set of training images, a query time that is independent of the database is most important to construct a scalable method. Min-Hash achieves constant query time, but has a comparatively low recall. Both of those properties are desirable in a large-scale setting, if the collisions are specific enough. With large-scale databases, there are many close neighbors and it becomes less important to find actual nearest neighbors. In this work, we will explore the potential of Min-Hash, as it is particularly interesting because of its geometric extension. It is a model-free method, so it can be easily updated with new data and can be employed as a black-box that supplies candidate neighbors quickly.

Chapter 3

Local Features for Retrieval and Categorization

This chapter explains the foundation of content-based image retrieval and categorization. We first give a brief overview of different types of features and then outline a method for finding robust features, called local affine covariant features. These features are used to obtain an image representation, which is suitable for applying text retrieval methods.

3.1 Types of Descriptors

The first step of image recognition, retrieval and categorization is usually finding a suitable representation of an image. A pixel-level representation is inapt for such high level tasks, because minor changes in perspective or illumination can affect every pixel in an image. For example, two images showing the same car at two different locations would have very different descriptors.

What we actually want is a descriptor which is invariant to translation of objects. This means we can easily find out if two images show the same object no matter where it is. Maybe we also want to allow for rotations, scale changes (a car could move away from the camera), or even slight appearance changes such as different lighting conditions (a car could look different at night). Not all of our requirements have to be served by the descriptor alone. In theory, many requirements can also be handled at a later stage as long as the descriptor does not eliminate the information we need. For example, local features discard the constellation of small patches in the image, but if we keep the positions of the patches we can still incorporate relative positions at a later stage.

There are two general types of image descriptors depending on the task and its requirements:

Global descriptors capture statistics about the whole image and summarize them in a way, such that our invariance requirements are met as well as possible.

Usually, those statistics describe color, shape, and texture in the image. The statistics can be collected for the whole image or for coarse subdivisions of the image, e.g. the image is divided into four regions of the same size and their descriptions are then concatenated.

A popular example is the GIST [OT01] descriptor. GIST initially contained novel global measures that were inspired by perception: naturalness, openness, roughness, expansion, ruggedness. It has been extended during the following years.

Another popular choice is the HOG descriptor [DT05]. It consists of overlapping histograms of oriented gradients. Even though it was developed for pedestrian detection, it is often applied to other tasks as well.

Local features describe small patches of an image, larger than pixels and smaller than entire objects, i.e. much smaller than subdivisions global descriptors summarize. Images are then represented as a collection of its parts.

There are different strategies for choosing the position and size of the patches.

- **Regular grid.** The simplest strategy is to extract patches on a regular grid with different scales of the image. The motivation is to make sure the representation captures all parts as well as their frequency in the collection.
- **Random.** Positions and scales are selected randomly. The idea is to get a description which is probably as good as the one constructed with regular sampling, but can save computational time by choosing a smaller number of regions.
- **Interest regions** are motivated by the idea to place the patches on points which are so distinct that it is possible to find the same point again, even if the perspective or lighting changes. This is very helpful if we want to detect correspondences between two images, because a repeatable localization of the regions improves the matching performance.

  The previous two methods are unbiased w.r.t. the distinctiveness of the patches, i.e. the relative frequency of parts in the collection of patches is approximately the same as the relative number of parts in the image, regardless of their distinctiveness. In contrast, interest regions create a collection of distinct parts of the image.

Research has shown that humans use both global and local appearance information for categorization [VSWB06], each yielding benefits for different types of categories. Global descriptors work well for near duplicate search, which makes them useful for image retrieval if the query object is contained in the database and is shown from a similar viewpoint [LWZ+08]. Yet, for more unrestrained image retrieval local features usually outperform global descriptors at the cost of higher computational complexity

[DKN08]. Local features perform particularly superior when utilized for matching small parts of images and can also be employed in a text retrieval-like manner to speed up the search [SZ03, PCI+07].

Local features loose their context, which is both an advantage and a disadvantage. As a consequence, local features are more robust to occlusion, cropping, and background clutter than global descriptors, while global descriptors vary if the background changes or parts of the object become invisible, the local features that are still on the object are relatively stable.

The omission of context of small regions also causes a disadvantage of the local feature approach: the spatial relationship between features is discarded. We will, however, keep the information of position and scale of features, to account for this shortcoming later.

3.2 Interest Regions

The aim of interest regions is to be reliably positioned regardless of viewpoint changes and noise. While noise is addressed by blurring of the image, repeatability is the main concern. We intend to find regions which can be found again as precisely as possible, even if the image is translated, rotated, or scaled.

The following sections describe a pipeline that finds interest regions. The first step introduces translation invariance by finding interest points. Building on that, we achieve scale invariance by running a scale detector at the interest points. The last detector selects the shape and orientation of the interest regions.

Homogeneous parts in an image are unsuitable for repeatable localization, because shifting a small window around in a homogeneous part does not change the content of the window. This is also the case for straight edges of an object; the window content does not change if it is shifted parallel to the edge or it is resized.

For stable and repeatable positioning of an interest region we need two edges that are not close to parallel, such as corners and sharp curvatures.

3.2.1 The Hessian Detector

The Hessian detector [Bea78] is based on the matrix of second derivatives, which is also called Hessian, of each point in an image. Its determinant gives information on the gradients and whether a point is a local extremum.

We want to compute derivatives on images, which is the same as computing the difference between two neighboring pixels. But photographs are usually subject to noise, which means that two neighboring pixels can have different colors even though the actual scene exhibits a uniformly colored region. The effect is a gradient at points without color changes in the real motive. Another source of “wrong” derivatives are compression artifacts which originate in the block-by-block separation of the image into small patches by the compression algorithms, e.g. the JPEG codec.

The remedy is a Gaussian blur which replaces every pixel by a weighted average of its neighborhood. Edges and wrong pixel colors are smeared and only strong and large color changes stay visible. This is achieved by the convolution of the image with a two-dimensional Gaussian function.

The starting point is an image in pixel representation. Usually all images are transformed into gray scale by averaging the red, green, and blue color channels, so they are reduced to only one channel. Given an image $I \in \mathbb{R}^{w \times h}$ of width w and height h, we denote its partial derivative at point $\mathbf{x}$ into direction d and d' using a Gaussian blur with variance σ^2 by $I_{dd'}(\mathbf{x}, \sigma)$. Then the Hessian and its determinant are defined as

$$\mathbf{H}(\mathbf{x}, \sigma) = \begin{bmatrix} I_{xx}(\mathbf{x}, \sigma) & I_{xy}(\mathbf{x}, \sigma) \\ I_{xy}(\mathbf{x}, \sigma) & I_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \in \mathbb{R}^{2 \times 2} \tag{3.1}$$

$$\det(\mathbf{H}(\mathbf{x}, \sigma)) = I_{xx}(\mathbf{x}, \sigma) I_{yy}(\mathbf{x}, \sigma) - I_{xy}(\mathbf{x}, \sigma)^2. \tag{3.2}$$

To illustrate what the derivative of an image means, consider the following example of $I_{xy}(\mathbf{x}, \sigma)$, which is the partial derivatives in x and y direction at the point $\mathbf{x} = (x, y)$:

$$I_{xy}(x, y, \sigma) = I_x(x, y, \sigma) - I_x(x, y-1, \sigma) \tag{3.3}$$

$$= (I(x, y, \sigma) - I(x-1, y, \sigma)) - (I(x, y-1, \sigma) - I(x-1, y-1, \sigma)) \tag{3.4}$$

where $I(x, y, \sigma)$ is the pixel value at point (x, y) in the blurred image I.

After calculating the determinant for all points in the image, we can perform a non-maximum suppression by sliding a 3 by 3 window over the image and dismissing all points that do not have a greater determinant than their neighbors. Remaining points that exceed a certain threshold are Hessian interest points.

The Hessian interest point detector is able to find corners regardless of their orientation and position by using the gradients present in the image. As a result, a translation of the image also translates its Hessian interest points and windows placed on them have translational invariant content. A rotation also rotates the interest points, however, the content of the regions around the interest points do rotate as well, as we have no means of detecting "the orientation" of a region yet.

3.2.2 The Hessian-Laplace Detector

So far we have achieved translation invariance. In the following we describe a method to also allow for scale changes. The solution for translation invariance was to select the position by the structure present in the image. Analogously, we will select the size of a circular region around an interest point by the structure of its neighborhood to achieve scale invariance.

The size of a region is commonly parametrized by a scale $\tilde{\sigma}$ from scale space $\mathbb{R}$, which could for instance parameterize a radius or the standard deviation for a Gaussian.

Note that, independent of how the scale is used at a later stage, the content of a region parametrized by $\tilde{\sigma}$ will have scale invariant content.

Scale invariant region detectors usually apply a *signature function* that calculates a signature score of the neighborhood of an interest point. For a larger scale, the signature function takes a larger neighborhood into account and calculates a different signature score. If this function is chosen to have one global maximum in the complete scale space, we can select this maximum as a reference point which is independent of the actual scale of the image.

The Hessian-Laplace detector [MS04] employs the scale-normalized Laplacian-of-Gaussian (LoG) as a signature function, which becomes maximal when covering a blob-like structure.

$$L(\mathbf{x},\sigma) \;=\; \sigma^2 \cdot (I_{xx}(\mathbf{x},\sigma) + I_{yy}(\mathbf{x},\sigma)) \tag{3.5}$$

After we obtained the Hessian interest points we search for the maximum of the LoG over scale space and use it as the size of the interest region.

$$\tilde{\sigma} \;=\; \operatorname*{argmax}_{\sigma} L(\mathbf{x},\sigma) \tag{3.6}$$

3.2.3 Affine Covariant Region Detection

The next step is to account for perspective changes. If we assume the interest region to lie on a planar object, it would be necessary to estimate out of plane rotations and perspective foreshortening. This procedure becomes too unstable, especially if we encounter small regions. A compromise is to model affine transformations which has proven to be sufficient for most cases. Affine transformations can be decomposed into translation, rotation, scaling, and sheering, so they can be visualized by ellipsoids. In the last sections we selected position and scale of circular regions, which we will now extend to rotated elliptic regions.

The procedure is said to be affine *covariant* because the affine region is chosen according to the image content. An affine transformation of the image causes the selected region to be transformed in the same way, so its content is *invariant*.

To understand the Hessian-Affine detector [MS04], consider the second moment matrix of a region selected by the Hessian-Laplace detector:

$$\mathbf{C}(\mathbf{x},\sigma,\tilde{\sigma}) = G(\mathbf{x},\tilde{\sigma}) \star_{\mathbf{x}} \begin{bmatrix} I_x(\mathbf{x},\sigma)^2 & I_x(\mathbf{x},\sigma)I_y(\mathbf{x},\sigma) \\ I_x(\mathbf{x},\sigma)I_y(\mathbf{x},\sigma) & I_y(\mathbf{x},\sigma)^2 \end{bmatrix} \tag{3.7}$$

The right-hand side of the convolution is the second moment matrix of one point $\mathbf{x}$ in the image capturing all combinations of first derivatives at that point. The left-hand side of the convolution is a Gaussian with the scale selected by the Hessian-Laplace detector. The convolution sums up the right-hand side for the neighborhood of $\mathbf{x}$, using the Gaussian for down-weighting points which are further away.

The second moment matrix can be visualized as an ellipse by plotting all points $\mathbf{x}$ for which $\mathbf{x}^{\mathrm{T}}\mathbf{C}\mathbf{x} = 1$. To see this, we can decompose the matrix $\mathbf{C}$ into rotation and

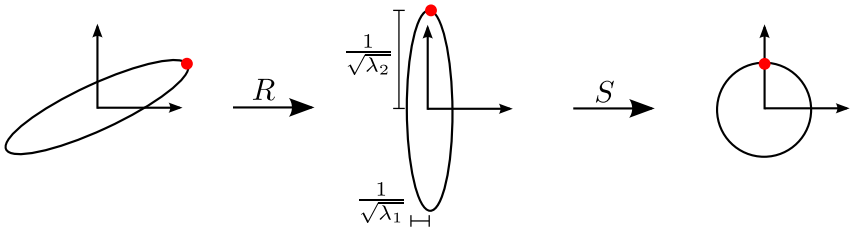


Figure 3.1: Visualization of a second moment matrix by a decomposition into rotation and scaling operations.

scaling operations using its eigenvalues λ_1 and λ_2:

$$\mathbf{C} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \tag{3.8}$$

$$= R^{\mathrm{T}} S^{\mathrm{T}} S R \tag{3.9}$$

$$\text{where} \quad S = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}. \tag{3.10}$$

R is a rotation matrix, which generally is orthogonal, i.e. $R^{-1} = R^{\mathrm{T}}$. S is a scaling matrix, which scales x and y components of vectors independently.

From Equation (3.9) we see that calculating

$$\mathbf{x}^{\mathrm{T}} \mathbf{C} \mathbf{x} = \mathbf{x}^{\mathrm{T}} R^{\mathrm{T}} S^{\mathrm{T}} S R \mathbf{x} \tag{3.11}$$

$$= (S R \mathbf{x})^{\mathrm{T}} \cdot (S R \mathbf{x}) \tag{3.12}$$

means applying a transformation to $\mathbf{x}$ and then computing the inner product to measure the distance from the origin. To understand what this decomposition means, we can look at the transformations applied to $\mathbf{x}$ before the inner product is taken. As depicted in Figure 3.1, we start with the ellipse that is defined by $\mathbf{x}^{\mathrm{T}} \mathbf{C} \mathbf{x} = 1$. First we rotate the ellipse by R so that its major axis lies on the y axis of the coordinate system. Then we scale the x and y dimension individually by $\sqrt{\lambda_1}$ and $\sqrt{\lambda_2}$ respectively using S in order to turn the axis aligned ellipse into a unit circle.

Now the Hessian-Affine detector finds an elliptic region around the interest point $\mathbf{x}$ depending on the second moment matrix $\mathbf{C}$ of the region the Hessian-Laplace detector selected. To obtain a stable result through view point changes, the affine region is morphed into a circle, $\mathbf{C}$ is recalculated, and this procedure is repeated until the changes of $\mathbf{C}$ are within small bounds.

In summary, the three preceding detectors form a pipeline, called Hessian-Affine interest region detector, that extracts affine covariant interest regions. The results are repeatable and robust within certain bounds of perspective changes.

3.3 Interest Region Descriptors

The Hessian-Affine interest region detector yields affine regions which subsequently have to be decoded into descriptors. For this task we have contrary requirements: on the one hand we need robustness against lighting changes and imprecise interest region detection, yet on the other hand we want regions with different content to be decoded distinguishably.

The most popular descriptor for recognition and categorization is the Scale Invariant Feature Transform (SIFT) descriptor. Some extensions and modifications have been proposed to improve its discriminative power or speed. For example, SURF [BTVG06] and GPUSURF [CVG08] have been developed to greatly speed up feature detection and description, which is particularly interesting for any real-time system.

An example for improved discriminative power is rgSIFT [vdSGS08], which adds color to the SIFT descriptor that otherwise operates on gray scale images. The SIFT algorithm is applied to the gray scale version of the image and to the red and green channels of a normalized RGB image, yielding a descriptor which is thrice as large as in standard SIFT.

3.3.1 The SIFT Descriptor

The Scale Invariant Feature Transform (SIFT) descriptor [Low04] achieves robustness to small translations by representing the gradients in the interest region by localized histograms.

First, the gradients of the interest region are calculated and weighted by a Gaussian, so points close to the center of the region have most weight. Those points are most likely to also be contained after a slight offset of the interest region after a viewpoint change, thus they shall have most influence on the descriptor.

The interest region is subdivided into a 4×4 grid where each cell's statistics are entered into one histogram with 8 bins. The gradient of each cell is sampled in a 4×4 grid and weighted votes are cast into the corresponding gradient histogram. To avoid discretization effects between the cells and between the histograms bins, trilinear interpolation is used to distribute votes simultaneously between the different cells' histograms and the histograms bins.

The histograms are concatenated into one $4\times4\times8 = 128$ dimensional vector, which is normalized and thresholded so its entries do not exceed 0.2, and finally normalized again. This post-processing is employed to normalize the contrast in the given region and avoid large impact of outliers. The elements of the final descriptor are usually scaled and rounded to integers between 0 and 127.

3.4 Bags of Visual Words

A simplifying assumption in natural language processing is to neglect the positions of the words in documents. The result is called "bag" because in contrast to a document it has no structure and in comparison to sets, elements may occur multiple times. Words are stemmed and replaced by their index in the vocabulary; the histogram of word indices is then a bag of words. The benefit of this approach is a terse representation, which is easier to handle.

Computer vision adapts this approach by building Bags of Visual Words (BoVWs). Instead of documents we have images and instead of words we have local feature descriptors. SIFT features, however, are too specific to be interpreted as words directly, so SIFT features have to be clustered to obtain a vocabulary. The new representation allows us to apply text retrieval algorithms to images, as for example *tf-idf search* in Video Google [SZ03] or min-Hash in [CPZ08].

3.4.1 Vocabulary Construction

The vocabulary is often called codebook, which refers to coding theory and, particularly, vector quantization for data compression. Vector quantization is exactly what we want to achieve: we want a huge universe of vectors (there are 128^{128} different SIFT descriptors) to be represented by a relatively small amount of vectors (e.g. less than 10^6).

Vocabularies are usually constructed by jointly clustering all features in the given data set. The resulting cluster centers form the vocabulary and every feature descriptor can then be represented by the index of the closest cluster center.

Methods for clustering can be distinguished based on whether they are supervised or not. Supervised clustering means that available class labels are integrated during the process, which enables discriminative training. Based on such class labels, we might want to optimize the visual vocabulary w.r.t. the class uncertainty of the contained words [MTJ07].

Unsupervised clustering is typically solved with k-means, an algorithm that minimizes the within-cluster sum of squared errors. Let $\mathcal{S} = \{S_1, ..., S_k\}$ be a partitioning of all data points $\mathbf{x}_1, ..., \mathbf{x}_N$ and μ_i be the mean of partition S_i, then k-means selects

$$\mathbf{S} = \operatorname*{argmin}_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|^2 \tag{3.13}$$

$$\text{where} \quad \mu_i = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x}. \tag{3.14}$$

This is achieved by alternating between updating the assignment of points to clusters and re-estimating the cluster centroids:

1. Before the first iteration select initial cluster centroids $m_i^{(1)}$ randomly from data points $\mathbf{x}_1, ..., \mathbf{x}_N$ and set $t = 1$.

2. Assign points to clusters by minimizing the distances to the cluster centroids:

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - m_i^{(t)} \right\| \leq \left\| \mathbf{x}_j - m_{i'}^{(t)} \right\| \quad \forall i' \in \{1, ..., k\} \right\} \tag{3.15}$$

3. Given the new partitioning, update the cluster centroids:

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x} \in S_i^{(t)}} \mathbf{x} \tag{3.16}$$

4. If the partitioning does not change anymore the algorithm has converged, otherwise increment t and continue from step 2. In practice this convergence takes a lot of iterations and the final steps hardly improve the results. Usually, a small percentage of changing cluster assignments suffice to consider the algorithm converged.

Clustering N points into k clusters with k-means unfortunately takes $\mathcal{O}(kN)$ distance calculations during the nearest neighbor search (step 2 above). Two popular approximations that make k-means clustering feasible are hierarchical k-means and approximate k-means [PCI+07].

Hierarchical k-means starts with clustering points into fewer clusters $k' = 10$ and then refines the clustering, by recursively applying hierarchical k-means to each of the clusters. With a recursion depth of d we get k'^d clusters. All k-means runs on depth d take $\mathcal{O}(Nk')$ distance calculations, so for small, fixed values of k' the overall number of distance calculations is reduced to $\mathcal{O}(Nk' \log_{k'}(k)) = \mathcal{O}(N \log(k))$.

Approximate k-means speeds up the clustering by using less distance calculations. This is accomplished by employing an approximate nearest neighbor search to reduce the number of nearest neighbor candidates at the cost of potentially missing true nearest neighbors. Using randomized k-d trees, the number of distance calculations is reduced to $\mathcal{O}(N \log(k))$.

While both methods have the same asymptotic time complexity, the hierarchical k-means does not optimize the k-means criterion (cf. Equation (3.13)) globally, but rather locally at each node of the hierarchy. Approximate k-means looses globally optimal guarantees by approximating the nearest neighbor search, yet it has proven to yield superior results [PCI+07].

3.4.1.1 Randomized k-d Trees

One method to reduce the number of distance calculations is using k-d trees, which partition space in a hierarchical fashion. A k-d tree is a binary tree that splits the space in each node by comparing one dimension of a point to a threshold.[1]

[1] Note that "k-d" refers to "k dimensional", not the number of clusters like in "k-means". We will continue to write k for the number of clusters and write D for the number of dimensions instead.

During construction, the dimension used for splitting is the dimension that exhibits the highest variance w.r.t. the points in that node. Typically, the median is applied for the threshold. The points in the node are then separated and recursively propagated to create the left and right subtree.

For the search of the nearest neighbor, we apply the test in the root of the tree and recursively continue the procedure in the left or right subtree, depending on the outcome of the test. This way, we try to find the closest points as early as possible. When the innermost function is finished and the recursion is "unrolled", we check if the closest neighbor we found so far is closer than the decision boundary that separates the subtrees. If the current closest neighbor is further than the decision boundary, we also need to descend into the other subtree to make sure we do not miss the real nearest neighbor.

This algorithm is guaranteed to find the true nearest neighbor, but the number of nearest neighbor candidates depends on how often the query point is too close to a decision boundary. As a rule of thumb, k-d trees work well when searching through many more than 2^D points, where D is the number of dimensions, e.g. $D = 128$ for SIFT. This condition is certainly not met for SIFT and all realistic values of k, so the k-d tree search would degenerate to a linear search [Hal04].

The root of the problem is that this type of approximation becomes worse for an increasing number of dimensions. A hypersphere of radius r, containing all points that are closer than r, is approximated by a hypercube. As the dimensionality increases, the volume of the sphere decreases relative to the volume of a cube. This means even if a query point falls into a hypercube that happens to tightly enclose the sphere we want to search through, the volume we have to search through in vain is growing and potentially contains a growing number of points. Every data structure that partitions space will inevitably fail to reduce the number of points we have to look at.

Fortunately, we do not need a solution for arbitrary dimensions. The idea to work with high dimensional spaces is to create multiple trees, which are all slightly different and create a smooth partitioning of space that avoids the quantization effects of a single tree. The tree construction process is randomized by choosing dimensions with a probability proportional to their variance and the median computation is randomized by working on a small sample of the data.

To search for the nearest neighbor of a query point we first descend all trees like for usual k-d trees, but the branches which are not taken are inserted into a shared priority queue, sorted by the distance between the query point and the corresponding decision hyperplane. When all trees have been descended once, we can obtain the most promising branch that was not taken from the top of the priority queue and descend the corresponding subtree. By bounding the number of times we take unused branches from the queue, we effectively bound the number of distance calculations.

3.4.2 Distance Measures

Thus far we extracted SIFT descriptors, clustered them, and summarized them in histograms, called BoVWs. Now we will examine distance measures between pairs of BoVWs.

Retrieval and categorization scenarios involve certain techniques to determine candidate images that qualify for being images that we are looking for — nearest neighbors. Those techniques are to speed up the search, so we can search through candidate images instead of all training images. Regardless of the details, we need distance measures between images to know which images are neighbors.

A standard distance is the Euclidean or L2 distance. Let I be the BoVW of an image, consisting of frequencies t^w for each Visual Word (VW) w.

$$\begin{aligned} \mathrm{dist}_{\mathrm{L2}}(I_1, I_2) &= \|I_1 - I_2\|_2 && (3.17) \\ &= \sqrt{\sum_w (t_1^w - t_2^w)^2} && (3.18) \end{aligned}$$

Using the Euclidean distance interprets the feature space as $\mathbb{R}^D$ and measures what we would intuitively call the diagonal or the beeline (of a D dimensional bee, though).

BoVWs, however, are histograms and the Euclidean distance is not interpretable in this context, as the diagonal between multiple dimensions does not make sense. The Manhattan or L1 distance sums up the difference in each dimension:

$$\begin{aligned} \mathrm{dist}_{\mathrm{L1}}(I_1, I_2) &= \|I_1 - I_2\|_1 && (3.19) \\ &= \sum_w |t_1^w - t_2^w| && (3.20) \end{aligned}$$

Note that the Euclidean and Manhattan distance converge for a decreasing number of dimensions.

Another choice that will prove useful later is the Intersection over Union (IoU) similarity, also called Jaccard similarity coefficient. It measures the overlap of two sets, normalized by the size of both sets. In the following, an image I is interpreted as a set of those words that appear at least once, i.e. $I = \{w : t^w \geq 1\}$, if it is used as a set.

$$\mathrm{sim}_{\mathrm{IoU}}(I_1, I_2) = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|} \tag{3.21}$$

The fact that the frequency of a word is omitted because sets are used instead is called *set assumption*. The IoU similarity is normalized and can be changed into a distance measure by flipping the sign:

$$\mathrm{dist}_{\mathrm{IoU}}(I_1, I_2) = 1 - \mathrm{sim}_{\mathrm{IoU}}(I_1, I_2) \tag{3.22}$$

The set assumption of the IoU similarity becomes problematic when the size of the vocabulary decreases. In very large vocabularies most BoVWs are sets anyway,

because the vast majority of VWs appear at most once. When the size of the vocabulary decreases, VWs tend to appear more than once. Information tends to be contained in *how often* a VW appears in an image, but not in *whether* a VW appears in an image.

The term frequency–inverse document frequency (tf-idf) weighting scheme is borrowed from text retrieval. Basically it considers a word more important if it appears often in a document, but less important if it appears in many documents. The tf-idf extension of the IoU similarity drops the set assumption and takes term frequencies t^w into account. It has also proven useful to down-weight VWs that appear in many different images, because they are less discriminative, even in the context of large vocabularies [PCI+07],

The traditional tf-idf score grows linearly with the term frequency and is weighted down logarithmically with the inverse document frequency. Let $\mathcal{I}$ denote a set of all images and t^w the number of occurrences of the VW w in image I. The tf-idf weighting of a VW w in an image I is defined as

$$\text{tf-idf}(w,I) = \frac{t^w}{|I|} \log \frac{|\mathcal{I}|}{\sum_{I \in \mathcal{I}} \delta(t_I^w > 0)} \tag{3.23}$$

where δ is the indicator function, which is 1 if the given expression is true and 0 otherwise.

The IoU similarity is extended to allow for the tf-idf weighting by defining the intersection and union of bags as the minimum and maximum of term frequencies respectively.

$$\text{sim}_{\text{tf-idf}}(I_1, I_2) = \frac{\sum_w d_w \min(t_1^w, t_2^w)}{\sum_w d_w \max(t_1^w, t_2^w)} \tag{3.24}$$

$$\text{where} \quad d_w = \log \frac{|\mathcal{I}|}{\sum_{I \in \mathcal{I}} \delta(t_I^w > 0)} \tag{3.25}$$

In some cases we will not be able to incorporate term frequencies, so we have to resort to the set assumption, but we can still keep the inverse document frequency weighting:

$$\text{sim}_{\text{idf}}(I_1, I_2) = \frac{\sum_{w \in I_1 \cap I_2} d_w}{\sum_{w \in I_1 \cup I_2} d_w} \tag{3.26}$$

Chapter 4

Locality Sensitive Hashing

This chapter discusses hashing techniques we employ for near-neighbor search. We give a brief introduction to Locality Sensitive Hashing (LSH) and then explain details of Min-Hash, which is the heart of all following extensions. These techniques build on the BoVW image representation from Chapter 3.

Hashing in the sense of the data structure "hash tables" is employed to access data with a specific key in constant time, i.e. without searching. Inspired by this algorithm, the idea of LSH is to hash data in such a way that similar data is hashed to the same cell of a hash table, so near neighbors can be found by a single look-up without doing any distance calculations. This is called LSH and has important applications for internet search engines, which otherwise could not cope with the increasing amount of information on the internet.

The problem boils down to finding hash functions that preserve the similarity of the data we want to search. This means similar items shall have a high probability of hashing to the same value, while dissimilar items have a low probability of being assigned to the same hash value.

Locality-sensitive functions usually appear in families, i.e. sets of functions that have the same properties. We can characterize families by the following common definition [RU11].

Definition 4.1. Let $d(x,y)$ be some distance measure, $d_1 < d_2$ two distances, and $p(f(x) = f(y))$ the probability of two items hashing to the same value. Then a family of hash functions $\mathcal{F}$ is said to be (d_1, d_2, p_1, p_2)-*sensitive* if for every $f \in \mathcal{F}$ the following conditions hold:

1. If $d(x,y) \leq d_1$, then $p(f(x) = f(y)) \geq p_1$ and

2. if $d(x,y) \geq d_2$, then $p(f(x) = f(y)) \leq p_2$.

Interesting families are obviously those for which $p_1 > p_2$. We call $p(f(x) = f(y))$ the *collision probability*.

We can insert item x into hash tables at position $f(x) \bmod m$, where m is the size of the hash table. This means the codomain of the hash function is not restricted to $\{0,...,m-1\}$. On the other hand, if some hash values are very unlikely because of the distribution of the input data, it is useful to choose m smaller than the maximal output of $f(x)$ to save memory. We will use closed addressing, which means every cell in a hash table contains a list of entries, because we have much more items in a table than m.

4.1 Min-Hash

Min-Hash (MH) is such a family of similarity-preserving hash functions. It was developed for near duplicate detection for documents and operates on sets. In natural language processing documents are represented as bags of words or as bags of tuples of words, whereas we use BoVWs for images (cf. Section 3.4).

To explain MH, assume that the bags are ordered. This is no restriction because bags are histograms and their contents are indexes of words in the vocabulary, so a natural order is imposed by the word indexes. An MH function is defined by a permutation σ of visual words; it rearranges the histogram according to σ and searches for the first word entry that is not zero. Put another way, the permutation defines the order in which we check if a VW is in the image. The first such word is the MH.

To make the procedure clearer, let us formalize it. Let $\mathcal{W}$ be the set of visual words, I be an image, represented as a BoVW, t^w specify how often a VW appears in image I, and $\sigma : \mathcal{W} \to \mathcal{W}$ be a permutation. We write $w \in I$ to select words that are present in the image, i.e. $t^w > 0$. Then the MH value is found by

$$\mathrm{mh}_\sigma(I) = \operatorname*{argmin}_{w \in I} \sigma(w). \tag{4.1}$$

The first thing to notice is that MH dismisses term frequencies. If a VW appears in an image, it does not matter how often. For this reason we have no hope to obtain a collision probability that is sensitive to term frequencies and we can interchange the terms "bag" and "set" for the standard MH analysis. However, MH has a simple but remarkable property, which is examined in the following theorem. The proof is taken from [RU11].

Theorem 4.1. *The collision probability of two BoVW through a randomly chosen MH function is equal to the IoU of the bags, as defined in Section 3.4.2:*

$$p(mh_\sigma(I_1) = mh_\sigma(I_2)) \;=\; sim_{IoU}(I_1, I_2) \tag{4.2}$$

Proof. We have two sets I_1 and I_2 and an MH function defined by a permutation σ that was chosen uniformly random. Let us define three sets of VWs:

1. X is the set of elements in both sets. Let $x = |X|$.

2. Y is the set of elements appearing in exactly one of both sets. Let $y = |Y|$.

3. Z is the set of elements that are in neither of the sets. Let $z = |Z|$.

Consider the histograms of I_1 and I_2 in which the words/bins are arranged by σ. The MH function looks for the first 1 in each of the histograms, so we look at bins of both histograms, starting at the beginning, in order to determine which cases influence the hash value.

1. We encounter a bin of type X before a bin of type Y. The word is contained in both I_1 and I_2, so $\text{mh}_\sigma(I_1) = \text{mh}_\sigma(I_2)$.

2. We encounter a bin of type Y before a bin of type X. The word appears only in one of both sets, so $\text{mh}_\sigma(I_1) \neq \text{mh}_\sigma(I_2)$.

3. We encounter a bin of type Z before any of the cases above. The hash values are not affected.

Only the first case results in the same hash value. Because in the third case we observe that type Z bins do not matter at all, we can conclude that "type X before type Y" is the necessary and sufficient condition to obtain the same hash value for both sets.

The probability of this condition to be true boils down to simple counting. There are $x + y$ different situations that matter to us, namely all bins of type X or Y, and which happens first is determined by σ. But only the bins of type X lead to a collision. Since the ordering of bins is uniformly random we can divide the number of positive outcomes by all possible outcomes, so

$$p(\text{mh}_\sigma(I_1) = \text{mh}_\sigma(I_2)) \;=\; \frac{x}{x+y}. \tag{4.3}$$

Now revisit the definition of X and Y. X are the words in both sets, so $X = I_1 \cap I_2$. Y are words in exactly one of both sets, so $X \cup Y = I_1 \cup I_2$ and because X and Y have no elements in common, $x + y = |I_1 \cup I_2|$. Overall, we see that the right-hand side of Equation (4.3) is exactly the IoU similarity $\text{sim}_{\text{IoU}}(I_1, I_2)$. □

The intuition behind Definition 4.1 is that a useful hash function has a higher probability to collide with similar items than with different items for a suitable choice of similarity. The previous theorem proves that MH accomplishes exactly that: the collision probability is actually *equal* to the IoU similarity. Since Definition 4.1 uses distance measures, we can use $d(x,y) = 1 - \text{sim}_{\text{IoU}}(x,y)$ as a distance measure, as mentioned in Section 3.4.2 to characterize MH:

> The family of MH functions is $(d_1, d_2, 1-d_1, 1-d_2)$-sensitive, for any distances d_1, d_2 where $0 \leq d_1 < d_2 \leq 1$.

4.1.1 Collision Probability Amplification

LSH is used to retrieve similar items. Due to the preceding collision probability analysis it becomes obvious that this is a probabilistic procedure. Consider a query item x and an item we want to retrieve y and assume their distance d_y is small but not zero, then y will be retrieved with probability $1-d_y$. It is possible that x and y do not collide in which case we call y a *false negative*. One the other hand, consider an item we do not want to retrieve z at distance $d_z < 1$. However large the distance might be, there still is a chance for x and z to collide. If this happens, z is said to be a *false positive*.

Hashing is usually employed if a set of data is simply too large to search through, which means that there are a few items we actually want to retrieve, whereas the vast majority should be filtered out. For a task with 1000 classes, where each class contains the same amount of images, we want to retrieve 0.1% of the images, the remaining 99.9% are potential false positives. For the sake of argument, assume the undesired images are far away from the query image at distance d_f, while the images we want to retrieve are at distance d_t. Assuming we have n images per class, we can calculate the expected number of true and false positives.

$$\mathbb{E}(\#\text{ true positives}) = n \cdot (1-d_t) \tag{4.4}$$

$$\mathbb{E}(\#\text{ false positives}) = 999 \cdot n \cdot (1-d_f) \tag{4.5}$$

If we want to expect no more false positives than true positives, we can constrain the distances d_f and d_t. Specifically for a distance d_t very close to zero, d_f already becomes very large, $d_f \geq 0.999$. From this example it becomes obvious that further modifications are necessary to make LSH useful.

So far, we used MH to summarize an image in a small *signature*, which consisted of one hash value. Instead of images, we can compare the signatures. Consider extending the signature by hash values of the same image through different MH functions. Then two signatures are equal if all s independent hash functions agree and the collision probability changes to

$$p_c(I_1, I_2) = \text{sim}_{\text{IoU}}(I_1, I_2)^s, \tag{4.6}$$

so the procedure becomes more specific. We call this group of independent hash functions *sketch*.

We can refine the collision probability further by using k independent sketches. While we have an AND association within a sketch, we choose an OR association between different sketches, so only one of the sketches suffices to collide in order for a pair of images to collide. The probability of a sketch not colliding is $1-\text{sim}^s$, so the probability of none of the sketches colliding is $(1-\text{sim}^s)^k$. With k sketches of size s we obtain the new collision probability

$$p_c(I_1, I_2) = 1-(1-\text{sim}_{\text{IoU}}(I_1, I_2)^s)^k \tag{4.7}$$

which is a sigmoid-like function, no matter how we change k and s.

> With this form of collision probability amplification it is possible to turn any (d_1, d_2, p_1, p_2)-sensitive hash family $\mathcal{F}$ into a $(d_1, d_2, 1-(1-p_1^s)^k, 1-(1-p_2^s)^k)$-sensitive hash family $\mathcal{F}'$, where each hash function in $\mathcal{F}'$ employs $s \cdot k$ hash functions from $\mathcal{F}$.

Intuitively, the advantage of a sigmoid-like function is that most of the change of the collision probability is compressed into a small change of the similarity. By increasing s, we can decrease p_2 close to zero while keeping p_1 relatively large. Conversely increasing k can increase p_1 significantly while keeping p_2 small.

We can think of the resulting procedure as a form of "probabilistic thresholding". The collision probability is either close to 0, close to 1, or quickly changing. The position of the steepest slope of the probability is approximately $\theta \approx (1/k)^{1/s}$. For similarities lower than θ the collisions are improbable, while higher similarities than θ lead to high collision probabilities. Sketches make the region of uncertainty around θ very small and allow us to influence the position of the similarity threshold through s and k.

The simplest implementation of hash tables employs an array of the same size as the vocabulary, as every VW can potentially become a hash value. Every hash table cell contains a list of entries, which are the images hashed to that cell. This way, the only ambiguity is caused by the MH function and no false positives are introduced by the implementation details of the hash table. We combine contents of cells of different hash tables within a sketch by intersection. This can be performed at run-time with the advantage that tables are stored for each hash function and can be combined into different sketch sizes without building new tables.

4.1.2 Word Weighting

MH revealed a collision probability equal to the IoU similarity of the corresponding images, but not all words are equally important. Usually, we want those words that carry the most information to have the most influence on the similarity between images. From an information theory point of view seldom events carry the most information, which leads to the idf part of tf-idf, as introduced in Section 3.4.2. In a categorization setting we can determine how discriminative words are and weight them accordingly. However the situation, the ability to weight words opens up a lot of new opportunities for improvement.

To derive word weighting according to [CPZ08] we have to revisit the definition of MH functions. In Section 4.1 we used permutations to find a MH value. Instead, we can understand a permutation of length n as an ordering defined by sorting n uniformly drawn random numbers, which leads to the following equivalent definition:

$$\mathrm{mh}_g(I) = \operatorname*{argmin}_{w \in I} g(w) \tag{4.8}$$

$$g(w) = r_w \qquad r_w \sim \mathrm{Un}(0,1) \tag{4.9}$$

The function $g : \mathcal{W} \to \mathbb{R}$ defines a permutation by assigning uniform random numbers between 0 and 1 to every VW $w \in \mathcal{W}$. Note that the drawing of random numbers is done once while defining g; g will always assign the same number to the same word. The new definition of the MH function uses the defining function g to determine which of the VWs in I is to be the hash value by choosing the word with the least image through g. The definition of mh_g is actually like in Section 4.1, but with the relaxation, that g does not have to be a permutation.

For now, assume that the desired word weights d_w are positive integers. We will later be able to drop this assumption. It is possible to incorporate integer weights by replicating every word w $d_w - 1$ times in the vocabulary and in every bag. The result are d_w distinct words, which represent the original word w. Let I'_i denote the modified set I_i with replicated, distinct words, then the IoU similarity changes as follows:

$$\begin{aligned} \text{sim}_{\text{IoU}}(I'_1, I'_2) &= \frac{|I'_1 \cap I'_2|}{|I'_1 \cup I'_2|} && (4.10) \\ &= \frac{\sum_{w \in I_1 \cap I_2} d_w}{\sum_{w \in I_1 \cup I_2} d_w} && (4.11) \end{aligned}$$

because each word $w \in I_i$ appears d_w times in both I_1 and I_2 and therefore also in the intersection and union.

Let $w^1, \ldots, w^{d_w}$ be the words in the new vocabulary $\mathcal{W}'$ that represent each word $w \in \mathcal{W}$ and $g'(w) : \mathcal{W}' \to \mathbb{R}$ a random hash function like in Equation (4.9), but over the new vocabulary $\mathcal{W}'$. Then we can use $g'(w)$ to create a random hash function $g(w)$ for $\mathcal{W}$, that generates the same MH values as $g'(w)$:

$$\begin{aligned} g(w) &= \min_{k \in \{1,\ldots,d_w\}} g'\left(w^k\right) && (4.12) \\ &= \min_{k \in \{1,\ldots,d_w\}} r_w^k \qquad r_w^k \sim \text{Un}(0,1) && (4.13) \end{aligned}$$

We can regard the minimization in Equation (4.13) as a random variable m_w, which has the cumulative distribution function (CDF)

$$\begin{aligned} p(m_w \leq a) &= p\left(\min_{k \in \{1,\ldots,d_w\}} r_w^k \leq a\right) && (4.14) \\ &= 1 - p\left(\min_{k \in \{1,\ldots,d_w\}} r_w^k > a\right) && (4.15) \\ &= 1 - p\left(r_w^1 > a, \ldots, r_w^{d_w} > a\right) && (4.16) \\ &= 1 - \prod_{k=1}^{d_w} p\left(r_w^k > a\right) && (4.17) \\ &= 1 - (1-a)^{d_w}. && (4.18) \end{aligned}$$

The calculation exploits the fact that in order for the minimum of a set of numbers to exceed a threshold a, all of those number have to be greater than a. Then we can use that r_w^k are independent random variables to factorize their joint in Equation (4.17).

Now we know which distribution we would like to draw random numbers from instead of $\mathrm{Un}(0,1)$ in order to achieve a word weighting. Fortunately, this distribution is easy to sample from, because its CDF is invertible. The inverse CDF method or inverse transform sampling states that sampling from m_w's distribution is the same as sampling a number from $\mathrm{Un}(0,1)$ and map it through m_w's inverse CDF.

$$m_w = \min_{k \in \{1,\dots,d_w\}} r_w^k \qquad r_w^k \sim \mathrm{Un}(0,1) \tag{4.19}$$

$$= 1 - \sqrt[d_w]{1 - r_w} \qquad r_w \sim \mathrm{Un}(0,1) \tag{4.20}$$

$$= 1 - \sqrt[d_w]{r_w} \qquad r_w \sim \mathrm{Un}(0,1) \tag{4.21}$$

The transformation from Equation (4.20) to Equation (4.21) is possible because if $x \sim \mathrm{Un}(0,1)$, then x and $1-x$ have the same distribution.

From Equation (4.21) we know how to sample random numbers for a function g, like in Equation (4.9), to use it for min-hashing. Since mh_g only uses the ordering of VWs defined by g, we can apply further simplifications without changing the result of mh_g. Valid simplifications have to preserve the ordering of the values of m_w, which is only the case for strictly increasing functions.

$$m_w < \tilde{m}_w \tag{4.22}$$

$$\Leftrightarrow \quad 1 - \sqrt[d_w]{r_w} < 1 - \sqrt[d_w]{\tilde{r}_w} \tag{4.23}$$

$$\Leftrightarrow \quad -(r_w)^{1/d_w} < -(\tilde{r}_w)^{1/d_w} \tag{4.24}$$

$$\Leftrightarrow \quad -\frac{\log r_w}{d_w} < -\frac{\log \tilde{r}_w}{d_w} \tag{4.25}$$

The transformation to Equation (4.25) seems invalid at first, because the logarithm is undefined for negative arguments. But since the transformation from x to $\log(x)$ preserves the ordering, the same is the case from $-x$ to $-\log(x)$. Technically, we apply a function $l(x) = -\log(-x)$, which is strictly monotonic for negative values of x, to both sides of Equation (4.24).

Finally, we are able to define a new random function that samples permutations from a distribution that respect word weights d_w as required. Note that the initial assumption that d_w is a positive integer is no longer required.

$$g(w) = \frac{-\log r_w}{d_w} \qquad r_w \sim \mathrm{Un}(0,1) \tag{4.26}$$

Intuitively, this means that words with a high weight probably are mapped to a small random number by g and therefore have a high probability of coming relatively early in the permutation used by mh_g. This solves the problem because the first "hit" in the BoVW is usually found early in the permutations.

We start from a minimum of d_w random variables in Equation (4.19), thus d_w has to be an integer. Then we replace the minimum of random variables by a parameterized

random variable with the same distribution. From the formula of the random variable in Equation (4.20) we see that d_w turned from a number of elements into a real valued parameter. Thus, we are able to drop the assumption of integer weights.

4.1.3 Term Frequency Weighting

As discussed in Section 4.1, the set assumption of MH does not distinguish between term frequencies greater than 1. Term frequencies, however, are an important property of BoVWs if the size of the vocabulary is relatively small. We incorporated word weights globally by modifying the sampling procedure of the permutations used for MH, but term frequencies are specific to individual BoVWs.

Term frequencies are integers, thus it is possible to represent multiple occurrences of a VWs w by distinct replications of that word $w^1, \ldots, w^{t_w}$ where t_w is the term frequency of w in the BoVW [CPZ08]. Like in the derivation of the word weights in Section 4.1.2, we create a new vocabulary $\mathcal{W}'$ from the old vocabulary $\mathcal{W}$ by duplicating individual VWs multiple times.

Let I_i a be BoVW consisting of term frequencies t_i^w for every VW w and N the number of all BoVWs. Then we can calculate the maximal frequency for every VW and define the new vocabulary:

$$\hat{t}^w = \max_{i \in \{1,\ldots,N\}} t_i^w \qquad \forall w \in \mathcal{W} \tag{4.27}$$

$$\mathcal{W}' = \left\{ w^k \,:\, w \in \mathcal{W},\, k \in \{1, \ldots, \hat{t}^w\} \right\} \tag{4.28}$$

$$\left|\mathcal{W}'\right| = \sum_{w \in \mathcal{W}} \hat{t}^w \tag{4.29}$$

where all w^k is distinct, so no two words have the same index.

Next, we create new BoVWs I_i' by replicating every BoVW I_i depending on its individual term frequencies using VWs from $\mathcal{W}'$.

$$I_i' = \left\{ w^k \,:\, w \in I_i,\, k \in \{1, \ldots, t_i^w\} \right\} \tag{4.30}$$

The key property of I_i' is that $w^k \in I_i'$ if and only if $t_i^w \geq k$, so we have t_i^w distinct instances of the word w. Furthermore those instances will be the same instances as in other BoVWs containing the VW w at least t_i^w times.

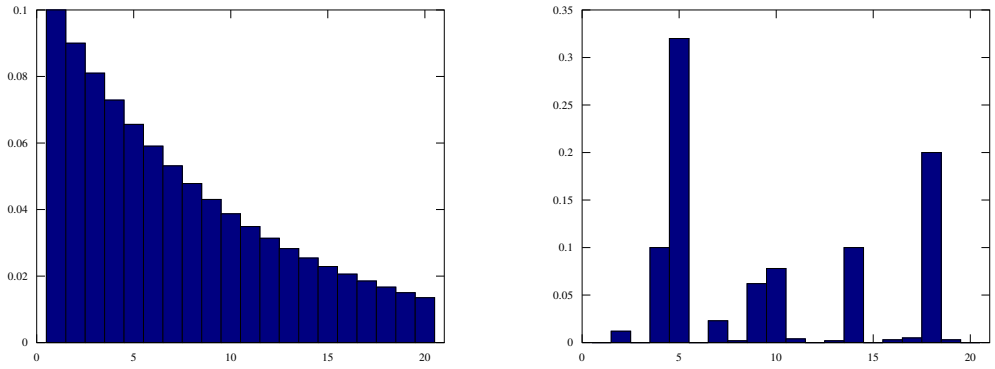
If we apply the standard MH procedure to the modified BoVWs, we obtain term frequency-sensitive collision probabilities.

$$\left|I_1' \cap I_2'\right| = \left|\left\{ w^k \,:\, w \in I_1 \cap I_2,\, k \in \{1, \ldots, \min\left(t_1^w, t_2^w\right)\} \right\}\right| \tag{4.31}$$

$$= \sum_{w \in \mathcal{W}} \min\left(t_1^w, t_2^w\right) \tag{4.32}$$

$$\left|I_1' \cup I_2'\right| = \sum_{w \in \mathcal{W}} \max\left(t_1^w, t_2^w\right) \tag{4.33}$$

$$\Rightarrow \quad \mathrm{sim}_{\mathrm{IoU}}(I_1', I_2') = \frac{\sum_{w \in \mathcal{W}} \min\left(t_1^w, t_2^w\right)}{\sum_{w \in \mathcal{W}} \max\left(t_1^w, t_2^w\right)} \tag{4.34}$$



(a) Distribution of i.i.d. data with $p_w = 0.1$.

(b) Exemplary distribution of real data.

Figure 4.1: Distribution of positions of MH values in the permutation, truncated after position 20.

Hence, the modification of the BoVWs caused the IoU similarity to be weighted by term frequencies and thus the collision probability of MH as well.

It is straightforward to combine word weighting and term frequency weighting. First, we apply term frequency weighting, which introduces new VWs and yields a larger vocabulary. Then, we apply word weighting by sampling permutations according Equation (4.26) where we use the weight d_w of VW w for all its replications w^k. Weighting every addend in Equation (4.34) with d_w yields the desired result:

$$\mathrm{sim}_{\mathrm{IoU}}(I'_1, I'_2) \quad = \quad \frac{\sum_{w \in \mathcal{W}} d_w \min\left(t_1^w, t_2^w\right)}{\sum_{w \in \mathcal{W}} d_w \max\left(t_1^w, t_2^w\right)} \tag{4.35}$$

4.1.4 Permutation Grouping

Permutation grouping is a method for reducing the number of collisions without reducing the number of true positives [BCI08]. The idea is to choose the grouping of permutations into sketches based on the distribution of the data.

MH chooses the first word in a BoVW according to a permutation σ. Consider the position l_i of the MH value in the permutation

$$l_i \quad = \quad \sigma^{-1}\left(\mathrm{mh}_\sigma(I_i)\right). \tag{4.36}$$

If we assume VWs to appear independently with a probability of p_w, we can estimate the distribution of l_i:

$$p(l_i = l) \quad = \quad (1 - p_w)^{l-1} p_w \tag{4.37}$$

The distribution of l_i turns out to be the geometric distribution, the discrete analogue of the exponential distribution, as shown in Figure 4.1(a).

In practice, however, we observe a much more unstructured distribution, for example like shown in Figure 4.1(b), even though the number of samples from the distribution is large enough. Clearly, the assumption of i.i.d. VWs does not hold. On the one hand, VWs generally do have different probabilities of occurrence. One the other hand, VWs certainly do not occur independently. It is possible to quantify the difference between the model and reality by calculating the entropy of both distributions.

Let N be the length of the permutation σ, which is also the size of the vocabulary $\mathcal{W}$.

$$H(l) \quad = \quad -\sum_{i=1}^{N} p(l=i)\ \log_2(p(l=i)) \tag{4.38}$$

In the case of $p = 0$, the limit of the expression is used:

$$\lim_{p \to 0+} p\ \log_2(p) \quad = \quad 0 \tag{4.39}$$

The entropy measures the uncertainty of a distribution; it is maximal for a uniform distribution and minimal if the result of the corresponding random experiment is predetermined. It is the expected information content of a random variable, where the information content of an event is the negative logarithm of its occurrence probability. Intuitively, in the case of the dual logarithm, we can understand the information content as the number of binary tests we have to make to tell an event apart from all others.

For i.i.d. generated data [BCI08] report an entropy of 6.7, while the entropy of the distribution of real data yields an entropy of approximately 4. This means, for real data, many items are hashed to the same value, specifically the data only occupies 6.3% of the hash table. This is called *clumping* and causes the disadvantage that most true positives are retrieved with many false positives.

When we use sketches, as discussed in Section 4.1.1, we have AND association between hash functions within a sketch. For hash tables this means taking the intersection of cells in the corresponding hash tables. An unfortunate combination of permutations leads to very large intersections and again many false positives. We can quantify this effect by using the mutual information of the distribution of the hash value positions l and l' in the permutations involved.

$$I(l,l') \quad = \quad H(l) - H(l|l') \ = \ H(l) + H(l') - H(l,l') \tag{4.40}$$

$$\quad = \quad \sum_{i=1}^{N} \sum_{i'=1}^{N} p(l=i, l'=i')\ \log_2 \left(\frac{p(l=i, l'=i')}{p(l=i)p(l'=i')} \right) \tag{4.41}$$

where $H(l|l')$ is the conditional entropy, which is the uncertainty about l, after the value of l' is known. The mutual information $I(l,l')$ measures how much knowing the value of l reduces the uncertainty of the value of l' or "how much information" l and l'

have in common. If l and l' are statistically independent, then we can factorize the joint $p(l=i,l'=i')=p(l=i)p(l'=i')$ and the addends in the sum become 0, so $I(l,l')=0$. Conversely, if l and l' are identical, the value of one random variable also determines the value of the other random variable, so $H(l|l')=0$ and thus $I(l,l')=H(l)=H(l')$. Those two extreme cases mark the maximum and minimum of the mutual information $0 \leq I(l,l') \leq \max(H(l),H(l'))$.

In the context of hash tables, the mutual information means how much information we gain about the content of a cell in one hash table, when we know the content of another hash table. This means high mutual information indicates large intersections between cells of two hash tables. In practice, the distribution of the mutual information over all pairs of hash functions reveals a long tail, which consists of unfortunate combinations of hash functions. Thus we can use mutual information as a selection criterion for groups of hash functions.

The selection procedure proposed in [BCI08] first calculates the mutual information of all pairs of hash functions and then greedily selects groups of hash functions that have low mutual information. Given we want to create k sketches of size s, let F be a set of hash functions, in which we will mark hash functions f as used by removing them from F, and G be a set of sketches, where each sketch $g \in G$ is a set of hash functions. Furthermore, the entropy and mutual information of the position of the MH values are the same as for the MH values, because there exists a bijection between the hash values and their position, namely σ. To avoid confusion, we will now use $H(f)$ instead of $H(l)$ for the entropy and $I(f,f')$ for mutual information.

1. Create sketches of size 1 by selecting the k hash functions with highest entropy and remove them from F.

   Formally, we repeat the following procedure k times.

$$\hat{f} := \operatorname*{argmax}_{f \in F} H(f) \tag{4.42}$$

$$G := G \cup \left\{\hat{f}\right\} \tag{4.43}$$

$$F := F \setminus \left\{\hat{f}\right\} \tag{4.44}$$

2. Select a sketch which is not full yet and add a hash function with minimal mutual information.

$$\hat{f},\, \hat{g} := \operatorname*{argmin}_{f \in F,\, g \in G:\, |g|<s} H(g,f) \tag{4.45}$$

   Then we add $\hat{f}$ to the sketch $\hat{g}$ and remove $\hat{f}$ from F.

3. Repeat step (2) until all sketches $g \in G$ have size s.

During the previous procedure, we encounter the problem of having to estimate the mutual information between a sketch g and a hash function f. This is trivial if there is only one hash function in g, but if there are more, we need to pool pairwise mutual information between f and hash functions in g. There are three common methods for accomplishing that.

1. Minimum pooling is the most aggressive method. It chooses that hash function to represent g which has the minimal mutual information with f.

$$I(g,f) = \min_{f' \in g} I(f',f) \tag{4.46}$$

2. Maximum pooling is the most conservative method. It uses the worst case, that hash function which has maximal mutual information with f.

$$I(g,f) = \max_{f' \in g} I(f',f) \tag{4.47}$$

3. Sum pooling is a compromise of the two previous methods. It takes all hash functions in g into account by using the sum of mutual information between f and all functions in g.

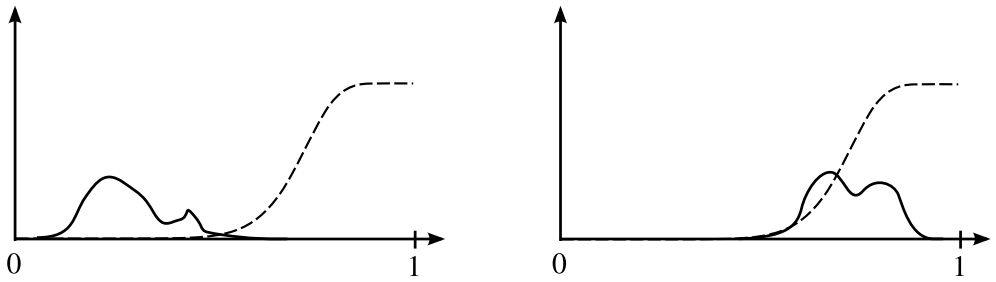
$$I(g,f) = \sum_{f' \in g} I(f',f) \tag{4.48}$$

This grouping algorithm does not yield an optimal result due to its greedy strategy, but the advantage is an asymptotic run-time behavior of $\mathcal{O}(|F|^2)$. Most notably, all effort is spent during construction time and there is no direct impact on the query time of the resulting hash tables. There is an indirect, beneficial influence on the query time, because the number of retrieved items is decreased.

4.1.5 Threshold Adaption

This section addresses a key difference between near duplicate detection and categorization. In near duplicate detection and sometimes also in image retrieval, we only want to retrieve those images that are above a certain similarity threshold, so we use LSH to retrieve candidate images and validate their similarity. In this setting, it is acceptable if the number of retrieved candidates is very low or zero. During categorization, however, it is very important that every query retrieves enough images, so its neighborhood is sufficiently represented to perform the actual categorization.

In Section 4.1.1 we discussed that the collision probability amplification is effectively a probabilistic thresholding. For near duplicate detection we can set that threshold according to how dissimilar we want to allow duplicates to be, in which case the variance in the number of collisions is deliberate. The number of collisions depends on the distribution of the similarity between the query images and the database.



(a) A distribution that yields few collisions.

(b) A distribution that yields almost the whole database.

Figure 4.2: Two exemplary distributions of similarities between a query and the training images. The dashed line indicates an exemplary collision probability as a function of the similarity (it is the same in both cases).

For example, Figure 4.2(a) shows the similarity distribution of a query, which is dissimilar from all training images. The dashed line shows an exemplary collision probability function with a threshold at approximately 0.7, while most training images are less similar than 0.7, so we expect very few collisions. For the other extreme, Figure 4.2(b) shows the similarity distribution of a query which is close to most images in the database. As most images have a similarity above the threshold, we expect to collide with almost the entire database. Both cases are bad for the categorization scenario. The first case does not yield enough neighbors to represent the neighborhood well, while the second case retrieves so many images that hashing does not pay off.

The idea is to adjust the threshold depending on the query's similarity distribution. It is easy to adjust the number of used sketches by successively querying the hash tables of a sketch and stopping if a certain number of collisions is retrieved. Increasing the number of sketches k effectively lowers the threshold until the desired number of collision is reached, which can be seen from the approximation of the threshold from Section 4.1.1: $\theta \approx (1/k)^{1/s}$.

The difference between standard MH and the threshold adaption is analogue to the difference between kernel density estimation and k-nearest neighbor (which is explained in greater detail in Section 4.3). Standard MH places a weighting window on the query which weights all images according to their distance from the query. The window has a fixed shape, which is called *bandwidth*, and sampling is performed from the product of the data distribution and the window. In contrast to that, threshold adaption increases the bandwidth until enough neighbors has been sampled.

4.2 Geometric Min-Hash

This section discusses a family of LSH functions called Geometric Min-Hash (GMH) [CPM07], which reintroduces geometric properties of VWs which are dismissed in the

BoVWs representation (cf. Section 3.4).

Spatial layout is an important property of images, especially for categorization and retrieval. Objects usually are spatially compact in the sense that parts which are close together are more likely to belong to the same object. Occlusions are objects which are closer to the point of view, so they are expected to be spatially consistent, too. The effect is that parts of an object we are looking for tend to disappear in groups that are close within the image.

The idea of GMH is to understand a sketch (cf. Section 4.1.1) as a region in an image. All hash values of one particular sketch are words inside of a certain region, in order to leverage the previous observations about spatial properties of objects and occlusions. This way GMH introduces spatial compactness into the hashing procedure, while standard MH produces hash values that are words all over the image.

The main problem consists of choosing the position and the size of the region we place a sketch on. We want to select the region in a repeatable way, i.e. we require a high probability of selecting the same position and size again, even through perspective changes. These requirements are similar to those discussed in Section 3.2.

Recall that MH values are VWs, which originate in interest regions that meet our requirements, so we let a VW define the region for a sketch. To select this word we use standard MH, so we obtain a high probability for selecting the same VW — and thus the same region — for images with a high IoU similarity.

GMH uses sketches as affine regions in an image, defined by the first MH function in the sketch, called *primary* hash function. For the subsequent MH functions in the same sketch, the BoVW is restricted to the primary feature's affine neighborhood as output by our affine covariant interest region detector and all other words are removed from the image description.

However, there is an important subtlety to the first MH calculation. We use a VW to determine a region in the image, therefore we require the VW to appear exactly once in the image, otherwise the procedure would be ambiguous. So, for the calculation of the primary hash value, we have to remove every VW that is not unique from the BoVW.

The MH calculation for all functions in the sketch except the first, also called *secondary* hash functions, are restricted to the affine neighborhood, but we have not yet defined how the neighborhood is to be determined in detail. We want to exclude those VWs from the procedure whose interest regions are too far away, both in image space and in scale space. The closeness in scale space is useful, because interest points can disappear, if they move either so far away that their distinctiveness vanishes or they can come so close that their appearance changes. As discussed above, we want to use the invariance of the used interest points. So, candidates for neighborhood region definitions are scaled versions of the affine interest regions. Additionally, we require interest points to have similar scale in order to be neighbors. The interest region is usually scaled up by a factor of 3, while the scale of neighboring interest points σ_{sec} is supposed to be within a factor of $\sqrt{2}$ to the primary VW with scale σ_{prim}:

$\sigma_{\text{prim}}/\sqrt{2} \leq \sigma_{\text{sec}} \leq \sqrt{2}\sigma_{\text{prim}}$.

So, the overall procedure is as follows:

1. Select a set of features U that are unique and have at least $\max(3, s)$ neighbors.
2. Apply the primary hash function to U to determine the primary hash value.
3. Select a set of features N, that are in the image and scale space neighborhood as defined above. Remove all features that appear more than once.
4. Apply the secondary hash functions to N.

The uniqueness constraint in step (3) is a different one than in step (1). While the items in U have to be unique throughout the whole image, the items in N are chosen to be unique in the neighborhood of the primary feature. The former is required for having a one-to-one mapping between MH values and local features in the image, the latter increases the distinctiveness of the secondary features.

For large vocabularies, the uniqueness constraint is not a significant restriction, as most words appear either once or not at all. A large vocabulary is the usual approach for retrieval, but for categorization it might be beneficial for the VWs to be less specific. In that case the uniqueness constraint becomes a serious concern, because it removes most of the features from the image description. We will later discuss a novel approach for incorporating words that appear more than once.

4.2.1 Analysis of the Collision Probability

In this section we provide deeper insight into the mechanics of GMH by analyzing the true positive rate. For further analysis, please refer to [CPM07].

For the sake of argument, assume that each VW appears exactly once per image. Consider the probability of two images depicting the same scene to collide. Each photograph shows the scene from a certain perspective and under certain conditions. Let ξ denote all parameters that affect the feature extraction process, e.g. lighting, viewing angle, and distance.

Now consider the standard MH procedure. We decompose the procedure in two random experiments. First we need the MH value to fall into the region that both images have in common. The probability $g(\xi)$ of that is the number of features in the overlapping region divided by the number of features in the union of both images. The second random experiment is the standard min-hashing which is restricted to the overlapping regions in both images. The probability $r(\xi)$ of a collision in the overlapping region is the IoU similarity. So we can decompose the probability P_{mh} of a true positive collision in a sketch of size s:

$$P_{\text{mh}} = g(\xi)^s \, r(\xi)^s \tag{4.49}$$

The GMH procedure is the same for the first hash function inside of a sketch. The following hashing is restricted to the affine neighborhood, so repeat the preceding decomposition on the neighborhoods in both images. Let $\gamma(\xi)$ be the fraction of features in the geometric overlap of both sketch regions and $\rho(\xi)$ the IoU similarity of the features inside of the geometric overlap. The probability P_{gmh} of a true positive collisions of GMH can be decomposed as follows:

$$P_{\text{gmh}} = g(\xi)\, r(\xi)\, \gamma(\xi)^{s-1}\, \rho(\xi)^{s-1} \tag{4.50}$$

The factors $g(\xi)$ and $\gamma(\xi)$ capture changes that affect the visibility like a change of perspective or occlusions. A change of appearance affects how well an interest region in the scene is detected and whether it is mapped to the same VW. These effects have an impact on the similarity and are captured in $r(\xi)$ and $\rho(\xi)$.

If we compare P_{mh} to P_{gmh}, we see that both methods have the factor $g(\xi)\, r(\xi)$ in common, which is the probability of one MH collision. MH and GMH need the first hash in a sketch to collide and this happens with equal probability. The difference lies in the hash functions 2 through s. GMH works on a small neighborhood and by this means replaces factor $g(\xi)^{s-1}\, r(\xi)^{s-1}$ by $\gamma(\xi)^{s-1}\, \rho(\xi)^{s-1}$. If the appearance change has a uniform effect on the entire overlap region, we expect $r(\xi) \approx \rho(\xi)$. Thus the difference boils down to $g(\xi)$ and $\gamma(\xi)$. If the primary hash functions matched a feature, the interest regions lie inside of the overlap of the two images. If we assume locally consistent occlusions, we have a high chance of both interest regions to coincide, because of the affine invariant interest region detection. This means $\gamma(\xi)$ is close to one and thus significantly larger than $g(\xi)$.

This analysis is a very simplified view of the collision probability. It just considers the case of a true positive image collision and even further assumes that there cannot be collisions between features that do not correspond to the same point in the scene, but it gives an idea about where the geometric aspect of GMH influences the collision probability.

4.2.2 Review of Min-Hash Extensions

Sections 4.1.1 to 4.1.5 give an insight into extensions of the MH method. This section reviews those extensions in the context of GMH to investigate whether they are still applicable.

Collision probability amplification is a universal LSH technique, which leads to the idea of sketches for MH. Sketches are the foundation for GMH, so they are already employed.

Word weighting turned out to be possible by modifying the sampling procedure for the permutations for MH. We can do the same for GMH, which has two effects. On the one hand, the selection of the sketch region is drawn towards heavily weighted VWs, on the other hand the similarity of regions is weighed according to the word weighting, analogously to MH.

Term frequency weighting. The standard GMH procedure operates on BoVWs that only contain VWs that appear once at most, so there are no term frequencies to respect.

For extensions of GMH that allow co-occurrences of VWs we can generally apply frequency weighting. The main problem is the renaming scheme for frequency weighting, as described in Section 4.1.3, and the fact that BoVWs of regions are subsets of BoVWs of the entire image. The scheme is based on the idea that a word w that occurs at least k times is represented by the same words $w^1, \ldots, w^k$ in all BoVWs, so the intersection and union have the correct size. Consider the case in which the sketch region contains only $k' < k$ of the VW occurrences. We would have to rename the words to make sure the sketch's BoVW contains $w^1, \ldots, w^{k'}$. This requires the hashing procedure to be aware of the frequency weighting renaming scheme and cannot be used as a black box for this extension.

Permutation grouping is aimed at a better utilization of the hash tables by reducing the number of retrieved items per sketch. This is efficiently done by minimizing the mutual information within sketches, which is only possible because pairs of hash functions can be examined without their sketch context. For GMH the distribution of hash values of a hash function depends on the primary hash function. As a consequence, secondary hash functions can no longer be analyzed without their sketch context.

It is still possible to simply ignore this problem and group according to the mutual information, but doing so disregards the principled derivation of the method.

Threshold adaption provides control of the number of retrieved collisions by querying sketch after sketch and stopping after a sufficient number of collisions is retrieved. This method effectively adapts the probabilistic threshold of the hashing procedure. It is straightforward to apply this technique to GMH.

In summary, collision probability amplification is already used and word weighting and threshold adaption are unproblematic. Term frequency weighting becomes complicated, while permutation grouping turns into a mere heuristic.

4.2.3 Multi-Region Geometric Min-Hash

Standard GMH only uses VWs that are unique in an image, which can be hazardous if the vocabulary is small and most words appear more frequently than once per image. The reason for this restriction is that we need to map an MH value to an interest region in the image, but MH does not distinguish between different occurrences. Allowing for multiple occurrences of a VW leads to a novel generalization of GMH.

If we allow co-occurrences of VWs, an MH value corresponds to multiple interest regions. During hashing of the training images and their insertion into hash tables, we

have no means to prioritize one region over another. Therefore, we have three general choices:

- From all regions, we just choose one randomly. This is a method that might work if the region disambiguation is rare, but generally it produces unrepeatable hash values and probably leads to false negatives.
- Merge all regions. The neighborhood of an MH value is defined to be the union of the neighborhoods of each of the VW's occurrences. The method yields repeatable results, but the more frequent the selected VW occurs in the image, the closer the method becomes to standard MH. This is a first, simple method, but high term frequencies will erode the specificity of GMH.
- Treat different regions as distinct instances of the same image, so that each region of one image can collide with every region of another image. The results will be repeatable and more specific, yet we have to insert each training image into the hash tables, potentially multiple times.

As the third method is the most involved one, this section concentrates on how to use all regions separately.

In order to allow collisions between all pairs of regions individually, we have to fulfill two properties. First, different regions from the same training image have to be entered into the hash tables in a way so that they are distinguishable. Second, a query image needs to query the hash tables with each of its region descriptions.

1. To make regions distinguishable, it is not sufficient to enter the image index multiple times into the hash tables.

   To see this, consider the following example: The MH value of an image I_1 appears twice in the image and we obtain two region descriptions $r_1 = (1,2)$ and $r_2 = (3,4)$ and a query image I_2 with just one region $s_1 = (1,3)$ for a certain sketch of size $s = 2$. Assume we insert image I_1 into the first hash table at positions 1 and 3 and into the second hash table at positions 2 and 4. Then a query with image I_2 looks up position 1 of the first hash table and position 3 of the second hash table, which both contain I_1, so both images collide even if none of their regions collide.

   This issue can be resolved by inserting image-region tuples, e.g. (I_1, r_1) and (I_1, r_2), or using an image indexing scheme that accounts for multiple regions, i.e. two different regions of the same image are assigned to different indexes.

2. During retrieval, we query every sketch's tables several times per image, once per region. The results of each region query can be merged using the union, provided that retrieved collisions are disambiguated at a later stage.

Employing both modifications leads to the perspective of colliding regions rather than images. This is a generalization of standard GMH, which represents an image by exactly one region per sketch.

4.2.4 Deleting Used Words

This section introduces a subtle method which has not been discussed in academic publications yet, that modifies the collision probability further to become more specific.

Standard GMH defines an image region using the primary hash function, restricts the image description to that region and subsequently applies all secondary hash functions to the region description. To increase the specificity of GMH collisions, it is possible to modify the similarity measure on the fly by deleting words from the region descriptions as they are used.

Consider two colliding images I_1 and I_2, i.e. one sketch of size s collides with the GMH signature $(\mathrm{mh}_1, \ldots, \mathrm{mh}_s)$ and both images have this signature. Let $r_1^{(1)}$ be the region description for the neighborhood of mh_1 in image I_1, and $r_2^{(1)}$ the same but in image I_2. Then by deleting used VWs, we obtain the following similarity:

$$\mathrm{sim}(r_1, r_2) = \prod_{i=2}^{s} \frac{\left| r_1^{(i)} \cap r_2^{(i)} \right|}{\left| r_1^{(i)} \cup r_2^{(i)} \right|} \tag{4.51}$$

$$\text{where} \quad r_k^{(i)} = r_k^{(i-1)} \setminus \{\mathrm{mh}_{i-1}\} \tag{4.52}$$

$$= r_k^{(1)} \setminus \{\mathrm{mh}_1, \ldots, \mathrm{mh}_{i-1}\} \tag{4.53}$$

Since a colliding MH value is contained in both involved BoVWs, deleting it from both BoVWs reduces the size of the union and the intersection by one. Hence we can rewrite Equation (4.51) to

$$\mathrm{sim}(r_1, r_2) = \prod_{i=2}^{s} \frac{\left| r_1^{(1)} \cap r_2^{(1)} \right| - (i-1)}{\left| r_1^{(1)} \cup r_2^{(1)} \right| - (i-1)}. \tag{4.54}$$

Each factor of the product is of the form I/U, where $I < U$. From that we can conclude, that $I/U > (I-1)/(U-1)$ and therefore the factors become smaller as i increases. This means the new similarity function is more restrictive than the IoU similarity.

Intuitively, we gain more information about the BoVWs as we force GMH to use words that have not been used in the current sketch before.

4.3 Re-ranking

So far, we discussed MH, several extensions and the similarity measure they approximate. The kind of approximation is not a continuous measure like the IoU similarity, but discrete collisions. Collisions between pairs of images occur with a probability proportional to a similarity measure, so hashing can determine near neighbor

candidates but is unable to rank them reliably. Furthermore, we have to expect to also retrieve false positives which are not actually close neighbors. Hence we need to postprocess the collisions in some way, provided that the hashing procedure retrieves enough candidates.

Because the quality of resulting neighbors is supposed to be high, we first retrieve more neighbor candidates by hashing than we actually need. Then we rank the candidates by their actual similarity, so we need to calculate the exact similarity. The aim of ranking is to truncate the ranking after a fixed number of candidates and this way dismissing as many false positives as possible, as we expect the false positive rate to increase with decreasing similarity. The remaining candidates can be used to perform categorization with any method that requires the local neighborhood.

A consistent design applies LSH with a collision probability proportional to a certain similarity measure. Then use the same similarity measure to rank all candidates and perform k-nearest neighbor (k-NN) classification. Nevertheless, a better similarity function during ranking would improve the performance, even if is not approximated by the hashing procedure.

In this case, k-NN classification boils down to a majority voting after all candidates below a certain rank are dismissed. There are other possibilities for carrying out the actual categorization, e.g. kernel methods or local versions of global methods as SVM k-NN [ZBMM06]. Since we want to investigate different hashing techniques, we want to decide for a simple classifier without introducing any more parameters.

Kernel methods and k-NN are both density estimation methods that can be employed for categorization by modeling the probability distributions of each individual class and subsequently applying the Bayes decision rule. Both methods are called non-parametric approaches, because they do not assume a certain parametric shape, such as for example a Gaussian.

The following derivation is taken from [Bis06]. Assume a point $\mathbf{x}$ was sampled from the distribution $p(\mathbf{x})$. Consider the possibility that $\mathbf{x}$ lies in a region $\mathcal{R}$ of volume V.

$$p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{y})\, d\mathbf{y} \tag{4.55}$$

$$\approx p(\mathbf{x})\, V \tag{4.56}$$

We can estimate the probability of $\mathbf{x} \in \mathcal{R}$ by sampling from the distribution $p(\mathbf{x})$. Assume we take N samples and the sample happens to fall into $\mathcal{R}$ exactly K times.

$$p(\mathbf{x} \in \mathcal{R}) = \frac{K}{N} \tag{4.57}$$

Combining Equations (4.56) and (4.57) yields

$$p(\mathbf{x}) \approx \frac{K}{NV} \tag{4.58}$$

If we want to estimate $p(\mathbf{x})$ from this insight, there are two possibilities. N is fixed by the given training data $\mathbf{t}_1, \ldots, \mathbf{t}_N$, so we can vary K and V. Either we fix V and count K, which leads to kernel methods, or we fix K and calculate V, which leads to k-NN.

Kernel methods place a weighted window $k(\mathbf{u})$ — a so-called kernel — on $\mathbf{x}$ and determine K by a weighted sum over the window. If the kernel is normalized the estimation can be written as follows.

$$\int k(\mathbf{u})\, d\mathbf{u} = 1 \tag{4.59}$$

$$K = \sum_{i=1}^{N} k(\mathbf{x}-\mathbf{t}_i) \tag{4.60}$$

$$\Rightarrow \quad p(\mathbf{x}) \approx \frac{1}{N}\sum_{i=1}^{N} k(\mathbf{x}-\mathbf{t}_i) \tag{4.61}$$

From this formulation we can see that this is the same as placing the kernel on all training instances and summing up the window contributions at point $\mathbf{x}$, when the kernel is symmetric $k(\mathbf{u}) = k(-\mathbf{u})$.

k-NN grows a volume around $\mathbf{x}$ that includes all points that are closer than a certain threshold, where the shape of the volume depends on the used distance function. The distance threshold is increased until K neighbors lie in the volume. Note that $k = K$ denotes the number of neighbors and has nothing to do with the number of sketches during MH.

After the volume containing K neighbors is determined, we use its volume V to estimate the density.

$$p(\mathbf{x}) \approx \frac{K}{NV} \tag{4.62}$$

The density estimated by k-NN is not a real distribution since the integral over $p(\mathbf{x})$ does not converge. However, we can still use it for categorization where it reveals a nice property.

Let $\mathcal{C}_j$ denote the jth category, N_j the number of samples from that class. We grow a volume V around the point $\mathbf{x}$, which is to be classified, until it contains exactly K points regardless of their class. Then we count the number of points K_j per class so we can estimate the class likelihood and priors.

$$p(\mathbf{x}|\mathcal{C}_j) \approx \frac{K_j}{N_j V} \tag{4.63}$$

$$p(\mathcal{C}_j) \approx \frac{N_j}{N} \tag{4.64}$$

Using the unconditional density estimation from Equation (4.62), we can use

Bayes' theorem to calculate the class posteriors.

$$p(\mathcal{C}_j|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_j)\, p(\mathcal{C}_j)}{p(\mathbf{x})} \tag{4.65}$$

$$\approx \frac{K_j}{N_j V} \frac{N_j}{N} \frac{NV}{K} \tag{4.66}$$

$$= \frac{K_j}{K} \tag{4.67}$$

With these posterior estimations, the Bayes decision rule boils down to a simple majority vote.

Both approaches have one parameter each. For kernel methods we have to choose a kernel that has a certain bandwidth, which defines how far points have to be to not have an influence anymore. For k-NN we have to choose k to determine among how many nearest neighbors the majority vote is performed. Choices for both parameters are based on certain assumptions about the data distribution.

We decide for k-NN to avoid the choice of a bandwidth, because the feature space of such a large categorization task is likely to be sampled very uneven.

Chapter 5

ImageNet Database

An integral part of this work is its applicability on large scale image collections. This chapter outlines the background of the used data and provides insight into the key peculiarities which make categorization of this data difficult. We also derive error measures based on the semantic structure of the categories of this database.

5.1 ImageNet

The ImageNet database[1] is one of the largest freely available image databases for visual categorization. Its semantic annotation is based on WordNet[2], a comprehensive collection of English words including nouns, verbs, adjectives, and adverbs. What makes it stand out is its rich annotation which was originally intended for machine translation of natural language.

All words in WordNet are grouped into sets of cognitive synonyms, called synonym sets (synsets). Most synsets are organized in a hierarchy, connecting semantically related synsets. Nouns for instance are structured in a from-general-to-particular and a from-whole-to-part manner, which means that synsets can have both multiple parents and multiple children. Statistics about the number of words and synsets of WordNet are summarized in Table 5.1.

The idea of ImageNet is to label a large number of images with WordNet's noun synsets to obtain a semantically meaningful distance measure between images through WordNet's hierarchy. The database is designed to resemble results acquired by an Internet search in terms of scale and image quality and diversity.

Large scale means that there are both a lot of categories (cf. Table 5.2) and a lot of images within each category. Having many categories makes the task realistic and difficult and stands contrary to traditional image datasets with merely a few classes, because the distance between categories decreases and there are more possibilities for misclassification.

[1] http://www.image-net.org/

[2] http://wordnet.princeton.edu/

Table 5.1: WordNet statistics.

Part of speech	Words	Synsets
Noun	117,798	82,115
Verb	11,529	13,767
Adjective	21,479	18,156
Adverb	4,481	3,621
Total	155,287	117,659

Image quality concerns technical problems such as the size and compression of images. There are very small (as small as 16×16 pixels) and very large images, with or without compression artifacts.

Image diversity expresses that appearances of two instances of the same category can be very different for reasons such as lighting conditions, viewpoint changes, occlusions, complex scenes, or dissimilar shapes. Examples for these properties can be found in Section 5.2.1. Some images also contain well more than one category, especially considering the number of synsets covered by ImageNet. Since images are only tagged with one synset, there are potential distractions everywhere in the image.

5.1.1 Construction

The construction of ImageNet greatly benefits from the additional information obtained from WordNet's synsets.

The synonyms in the synsets and their translation to other languages, e.g. Chinese, Spanish, Dutch and Italian, are used to query image search engines. This yields a large amount of candidate images, containing many false positives.

Verification of the candidate images is manually done by humans using Amazon Mechanical Turk (AMT). AMT is an online platform where users perform tasks for money that are not easily done by a computer because they require human intelligence. In this case, AMT users are provided with a set of candidate images, along with the synset definition and a link to the corresponding Wikipedia article and are asked to filter the shown images.

Even though or rather because the annotation is done by humans, the results are subject to noise — none of the annotators are experts. So every image has to be reviewed by multiple users until a certain confidence is reached, which depends on the category, because the highest obtainable confidence varies greatly with the category.

Table 5.2: ImageNet statistics. Some of the largest high level synsets, along with the number of contained synsets (children) and the average number of images per contained synset.

High Level Synset	Contained Synsets	Images per synset
animal	3,822	732
device	2,385	675
food	1,495	670
mammal	1,138	821
plant	1,666	600
structure	1,239	763
person	2,035	468

Table 5.3: Statistics about the ILSVRC2010 corpus.

# of images	1,461,206
# of categories	1,000
# of synsets in subhierarchy	1,676

5.1.2 Statistics

ImageNet currently contains 12,184,113 images, covers 17,624 of WordNet's noun synsets (cf. Table 5.2), and 657,827 of the images have bounding box annotations. The aim is to provide 1,000 images on average for each of WordNet's 100,000 synsets, also including those which are not nouns.

Specific examples illustrating the nature of ImageNet can be found in Section 5.2.1.

5.2 Large Scale Visual Recognition Challenge 2010

The ImageNet Large Scale Visual Recognition Challenge 2010 (ILSVRC2010) was held together with the PASCAL Visual Object Classes Challenge 2010 (VOC2010). It uses part of the ImageNet data, 1.2 million images labeled with 1,000 categories (Tables 5.3, 5.4). The task was to predict the categories of 150,000 unlabeled images, given the training set annotation and the WordNet hierarchy.

The categories are synsets chosen from the low levels of the WordNet hierarchy (not necessarily leaves) in a way that one category is not an ancestor of another. It was, however, encouraged to use the relevant subhierarchy of WordNet, e.g. all category synsets and their ancestors up to the root, in which the category synsets actually are leaves.

Table 5.4: Size and partitioning of the ILSVRC2010 corpus. The cell for instances per training category shows min/max/mean.

Partition	Size	Instances per category
Training	1,261,206	668/3047/1261.4
Validation	50,000	50
Testing	150,000	150
Total	1,461,206	

Table 5.5: Number of categories within concepts which could already be used as categories.

General concepts	Categories contained
Snake	13
Monkey	19
Dog	36
Flower	88

5.2.1 Examples

The data of ILSVRC2010 is chosen in a way that it inherits properties of the ImageNet database, mentioned in Section 5.1. This section explains the main problems which are to be expected using some example images.

High variance within, yet small distance between categories. With a thousand categories, ILSVRC2010 has considerably less concepts to learn than ImageNet, but the number is still high enough to cause the distance between different categories to become smaller while keeping a high variance within categories.

Figure 5.1 provides an illustration of this phenomenon: 5.1(a) and 5.1(b) both show candy shops, from the inside and the outside respectively. Here, the difference in perspective changes the appearance of quite possibly the same object dramatically. In the first image it is possible to recognize the candy shop by the amount of candies and how they are arranged, while in the second image it is actually necessary to be able to read, to figure out the function of the shack. This pair of images illustrates two images of the same category having a high semantic similarity, yet they have a low visual similarity.

On the other hand Images 5.1(c) and 5.1(d) show two different categories — a celandine poppy and a lesser celandine respectively — so their semantic similarity is lower than in the previous example, but their visual similarity is much higher. In fact they look similar enough that the AMT annotations are often wrong: lesser celandines are often labeled as celandine poppies and vice versa.

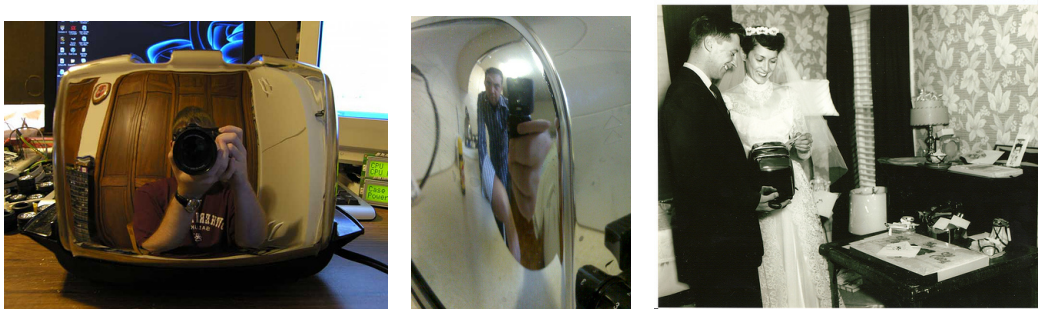
(a) Category “candy store” can only be inferred from individual parts and their constellation.

(b) Category “candy store” only recognizable from text on the shop sign.

(c) Category “celandine poppy”.

(d) Category “lesser celandine”.

Figure 5.1: (a, b) Two instances from the same category which look very different. (c, d) Two images from different categories which look more alike than the two candy stores. The visual similarity is higher, but the semantic similarity is lower.



(a) (b) (c)

Figure 5.2: Example images from the category “toaster”.

The number of categories in a categorization task affects how the data covers the semantic and visual domains. One effect of increasing the number of categories is that the data base covers new semantic and visual areas. Another more severe implication arises by subdividing categories: the categories move closer, both semantically and visually. Some examples of the latter effect are shown in Table 5.5. The left column shows perfectly valid categories humans use in everyday life and the right column shows how many subcategories are actually contained in the ILSVRC2010 dataset.

Images that are not photo-realistic increase the within-variance of categories artificially. Examples are paintings, rendered computer graphic images, or miniatures (e.g. Figure 5.3(b)).

Requirement of different levels of understanding. For categorization, the common task we perform as humans is localizing an object in an image and then classify it, which is basically working with a few localized cues.

One step further would be connecting a lot of local cues to gain a global understanding of the image, e.g. for recognizing the candy shop in Figure 5.1(a). You can see this as recognizing a scene as a whole by just using frequencies of its parts.

The other candy shop in Figure 5.1(b) is not recognizable by its appearance at all, but you have to understand that there is a sign on top of the shack and be able to read it.

Part of working with realistic and unfiltered images means dropping the assumption that objects are usually depicted centered and that they are well visible. An interesting example for this is provided by the “toaster” category. A closeup photography of a reflecting toaster, such as in Figure 5.2(b), renders the actual object virtually invisible. The main cues for recognizing the toaster are patterns in the distortion of the reflected scene.

(a) A tandem bicycle. (b) A forklift.

Figure 5.3: (a) An image from the category “tandem bicycle”. The object is not visible. (b) An image from the category “forklift”. It is not actually a real forklift but a LEGO miniature.

Another interesting example is the picture of two people on a tandem bicycle in Figure 5.3(a) where the bicycle itself is not shown. We can tell from the clothing that those people are probably cycling, but to classify the image as “tandem bicycle” we have to judge by the distance between the two people and their pose that they are sitting on the same bike.

The last two examples require real understanding to recognize something which is not there because none of the parts are visible.

Figure 5.4 shows three different images from the category “costume”. The objects are nicely centered, but the appearance is obviously diverse. In each case, we see people wearing something. But to come to the conclusion that it is a costume, you need to be able to assess the situation and come to the conclusion that the dress is unusual or inappropriate in some way; this can be a woman wearing a mustache, a black Roman at a party (Figure 5.4(b)) or a drunk female craftsman (Figure 5.4(c)). So in these cases after understanding the scene we had to judge that the details which are off are actually significant, so significant they make up the category of the image.

Multiple categories per image. Realistic image collections, in the sense of not being artificially constructed but resembling appearances of most usual images in the internet, exhibit a new quality of distracting objects. Multiple objects in an image are the regular case and now that we have many categories, some of those objects will fall into categories we are using as target labels. Figure 5.2(c) for instance shows an image from the category “toaster”, you can see several objects belonging to other categories, for example “lamp”, “lampshade”, “window shade”, and “suit”.

Background clutter and occlusions. Objects in an image that are not the objects we

(a) (b) (c)

Figure 5.4: Example images from the category “costume”.

are looking for (the annotation of the image) are called *clutter* or *stuff*. Clutter makes the task difficult because it exhibits structure we are not interested in, which distracts algorithms from the object.

Occlusions are a challenge because the appearance of an object changes from a global perspective. From a perspective of local descriptors parts of an object disappear entirely.

Some databases explicitly only contain clean images without clutter or occlusions. During the construction of ImageNet, namely in the AMT filtering, people were encouraged to accept images with occlusions and clutter.

Noisy image labels. Some categories are very hard for laymen to delimit. During labeling in AMT, the task was not to sort images into the right categories, but to filter out false positives for each category separately. Annotators were aided with a description of the category, but they have no way to determine the details of a category that distinguish it from closely related categories. As a consequence, images that belong to another category are sometimes not filtered out because they look very similar. For example, they learned that a celandine poppy is a yellow flower, not that it has four petals, two falling sepals, many stamens and a single knobby stigma, so they cannot distinguish it from a lesser celandine.

If the annotators are no experts, are paid per annotated image and some of them decide it is not worth spending a lot of time looking at the definition and the images in question, the construction system (cf. Section 5.1.1) concludes that this category is very hard and allows a low confidence score per image. The confidence score may even stay high if all annotators decide to allow lesser celandines in the celandine poppy category, for they are both yellow.

Celandine poppies are a good example for wrong labels. The category can be browsed online at `http://www.image-net.org/synset?wnid=n11908846` (unfortunately that is the current ImageNet category, not the training images for ILSVRC2010). It is possible to pinpoint false labels just by comparing the number of leaves and their shape.

5.2.2 Error Measures

For ILSVRC2010, there are a couple of error metrics taking account of the properties of the image collection and the structure of the image categories. There is a *flat error* and a *hierarchical error* for the basic error calculation, and there are *top 1* and *top 5* variants for both of them, to account for the diversity of the images.

5.2.2.1 Flat Error

The flat error is the commonly known unweighted error rate, hence the name. It is the number of misclassifications divided by the number of test images. To unify the notation with the following hierarchical error, we use a cost matrix (confusion matrix) filled with ones, except for the diagonal, which is zero.

To formalize this, let $I \in \mathcal{I}$ be an image from the test set $\mathcal{I}$ of size $N = |\mathcal{I}|$, c_I its annotated category and k_I the prediction of image I. Then the cost matrix C^{flat} would be

$$C^{\text{flat}}_{c,k} = \delta(c \neq k) = \begin{cases} 0 & \text{if } c = k \\ 1 & \text{else} \end{cases}. \tag{5.1}$$

So we can write the flat error as

$$\text{Err}^{\text{flat}}(\mathcal{I}) = \frac{1}{N} \sum_{I \in \mathcal{I}} C^{\text{flat}}_{c_I,k_I}. \tag{5.2}$$

5.2.2.2 Hierarchical Error

Since the categories in the dataset are structured into a hierarchy, it is possible to use that hierarchy to measure the semantic difference between categories and weight the error accordingly.

The category c_I of an image I will be neither a child nor a parent of the predicted category k_I (this is how the labels in ILSVRC2010 are chosen), so they are either the same or live in different "subhierarchies". We can determine the lowest common ancestor (LCA) w.r.t. the WordNet hierarchy and conclude how general or specific the concept is, which has both c_I and k_I in common. The depths of synsets in the hierarchy are already designed to express the different levels of generality, so we can use these depths as a metric:

$$C^{\text{hier}}_{c,k} = \begin{cases} 0 & \text{if } c = k \\ \text{height of LCA of } c \text{ and } k & \text{else} \end{cases} \tag{5.3}$$

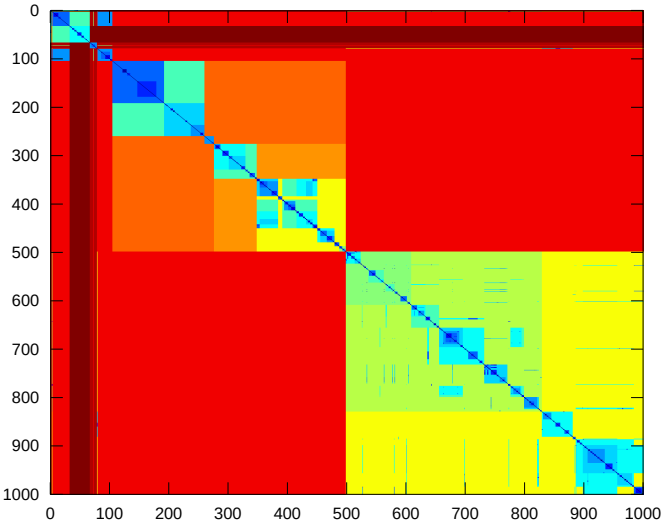


Figure 5.5: A visualization of the hierarchical cost matrix of ILSVRC. x and y axes are categories (in the same order). Red means high misclassification cost and blue means low misclassification cost.

We can use the hierarchical cost matrix C^{hier} to calculate the hierarchical error analogously to Eq. (5.1):

$$\text{Err}^{\text{hier}}(\mathcal{I}) = \frac{1}{N} \sum_{I \in \mathcal{I}} C^{\text{hier}}_{c_I, k_I} \tag{5.4}$$

Figure 5.5 shows a visualization of C^{hier} as a heat map. Blocks of cold colors mark categories with close semantic concepts, because they share a low common ancestor. Those blocks form a subhierarchy, which is some sort of semantic concept. You can also see how different one category is from most of the other categories by looking at one slice through the diagram.

5.2.2.3 Top 5 Error

The error functions Err_{flat} and Err_{hier} are called top 1 errors because only one prediction per image is allowed.

Considering that there often are several objects in an image that belong to different categories and there are also annotation mistakes, the idea of the top 5 error is to allow a prediction of five different categories per image. The top 5 error function uses only that prediction of five, which causes the least cost per image.

For an arbitrary cost matrix C and a set of predictions K_I per image I, where $|K_I| \leq n$ we can calculate the top n error in the following way:

$$\text{Err}_{\text{top } n}(\mathcal{I}) = \frac{1}{N} \sum_{I \in \mathcal{I}} \min_{k_I \in K_I} C_{c_I, k_I} \tag{5.5}$$

Table 5.6: Statistics about the tiny set. Relative numbers are in relation to the complete ILSVRC2010 dataset.

	Absolute	Relative
Classes	100	10%
Training Images p. Class	ca. 120	10%
Evaluation Images p. Class	10	20%
Total Number of Images	13,419	1%

Overall, there are four different error functions obtained by the combinations of top 1 and top 5 scoring and the weighted and uniform cost matrices.

5.2.3 The *tiny* Set

During exploratory work it is important to be able to evaluate choice of parameters and new ideas quickly. To this end, we built the *tiny set* to be a hundredth of the size of the complete ILSVRC2010 dataset.

We sampled 100 classes as follows: we chose random groups of 100 classes from ILSVRC2010 and selected the group that maximizes the pairwise hierarchical distance within the group. This way, we make sure that we get samples from all semantic areas and do not concentrate on one subhierarchy. On the downside, the classes will become easier to distinguish and the performance we increase, which we have to take into account when evaluating on the tiny set.

For each class we randomly chose 10% of the ILSVRC2010 training images to be the training set for the tiny set. Evaluation is done on the first 10 images of each class in the evaluation set of ILSVRC2010.

Statistics about the size of the tiny set and a comparison to the size of the original ILSVRC2010 set are shown in Table 5.6.

Chapter 6

Pipeline Design

This chapter serves a twofold purpose. On the one hand, it clarifies how the components from Chapters 3 and 4 are assembled into one pipeline. Important aspects are the order of methods and which parts are optional. On the other hand, the scale of the database constitutes an important aspect of the implementation and the design. Building on the deeper understanding of the pipeline, we will discuss how to handle that data with the given resources.

6.1 Pipeline Overview

In this section, we will investigate each of the components more closely, in order to understand nuances and implications of every step. This is crucial to study memory and run-time usage. We will discuss trade-offs later on.

A conceptual high-level view of the pipeline can be seen in Figure 6.1. First, all images are represented as BoVWs, which is covered in Chapter 3. Based on the BoVW representation, we hash all training images and insert them into hash tables. Then we hash all test images and query the hash tables for collisions, which we use as near neighbor candidates.

Up to this point in the pipeline we did a generic near neighbor search. There are a variety of methods that can employ it as a black box, but for evaluation of the pipeline

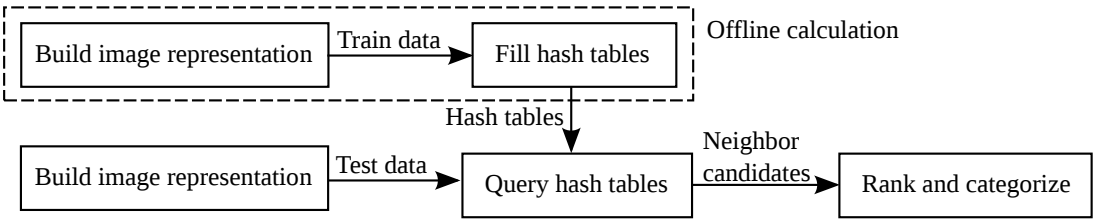


Figure 6.1: A coarse pipeline overview. The dashed rectangle of the pipeline only contains offline calculations that do not have to be carried out for every query.

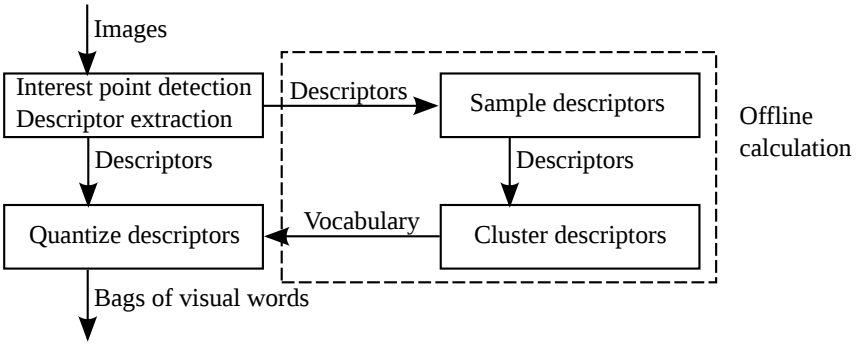


Figure 6.2: An overview of the pipeline that builds BoVWs from images. The dashed rectangle denotes the part of the pipeline which can be performed offline.

we choose a simple possibility, k-NN classification.

Another important aspect of this pipeline is that part of it only involves offline calculation. The parts of a pipeline are usually classified into offline and query-time calculations. Offline calculations are performed once, while query-time calculations have to be carried out every time a query is done. For a fast system it is beneficial to shift as much computation time as possible into the offline phase.

The offline part of the pipeline is marked by a dashed rectangle in Figure 6.1. It includes building the image representation of the training images, hashing them and inserting them into hash tables. For every query image, we have to extract the BoVW of that image, hash it, and query the hash tables. The resulting collisions are used for categorization.

In general, we want the offline part to handle as much data as possible in as little time as possible. If a system is supposed to work in real-time, we would like a query to have minimal latency until the categorization decision is made. Yet for experiments in which we simply categorize a lot of images in a row, we simply have the same requirement for the query-time part of the pipeline as for the offline part: finish the task in a short amount of wall-clock time.

6.1.1 Feature Extraction

This part of the pipeline starts from plain images and calculates BoVWs for each of them. It is a component of both the offline and query-time part of the pipeline as shown in Figure 6.1, but an expensive part of it can be executed offline.

The interest region detectors and descriptor extractor have to be applied to every image. The extraction of different images is independent, so this step is easily parallelized. The resulting SIFT features are extracted, as described in Sections 3.2 and 3.3.1, and saved along with their position and affine interest region.

Before the SIFT features are clustered, we sample a small set of features to speed up the clustering process. There are 1.2 million images with approximately 1.000 features each, so using them all can reach memory and run-time limits. Usually about

10 million SIFT features are sampled, which can be done by sampling a small portion of features from every image.

A vocabulary is constructed using the k-means algorithm and randomized k-d trees, as discussed in Section 3.4.1. The procedure jointly clusters the whole feature sample, which is not easily parallelized in a standard queueing system. Multithreading, however, is simple to implement using shared memory. Most of the procedure's run-time is spent on distance calculation, so it benefits greatly from vectorization and multithreading.

The clustering is sped up by using parameters which cause the randomized k-d trees to have a relatively high false nearest neighbor rate. The resulting clustering can usually still be sufficient, especially for low vocabulary sizes.

For SIFT feature quantization we select parameters that yield a much higher precision. The quantization can be prepared offline by training a forest of randomized k-d trees on the vocabulary. This is particularly useful since quantization is also needed during query-time.

The clustering and sampling is only done once and offline. Feature extraction and quantization are part of every query procedure, but are also used to construct BoVWs for the training images.

6.1.2 Hash Table Construction

The entire hash table construction is performed offline. There are three independent components in this part of the pipeline, which are extensions of the standard MH and GMH procedure.

The core tasks are to sample permutations to define MH functions, hash the training data and insert them into hash tables, which is covered in Section 4.1. An overview is shown in Figure 6.3, where optional parts are dashed. The hashing procedure can be parallelized, as hashing of different BoVWs is independent.

Hash table construction turns out to be hard to parallelize. For filling one table, we need to read the hash values of every training image, and for entering one training image, we need to write to every table, hence both dimensions are not inherently parallel. A map-shuffle-reduce framework would be able to solve the problem, whereas a standard queueing system has no means for that and puts considerable load on the central file server.

The term frequency weighting extension (cf. Section 4.1.3) has to be applied to each BoVW of training and test set. The size of the vocabulary changes, since VWs are replicated, so this is the first operation that is applied to BoVWs. The operation on different BoVWs is not independent if the mapping from old VWs to new VWs is unknown, because the method is based on the fact that the same word-frequency pairs are always mapped to the same new VW. Since we do not build an on-line system that actually has to work with unknown queries, this is not a concern in this work, as we can apply the mapping while defining it with one pass over all training and test BoVWs. In an on-line system, we would have to apply the method to the training

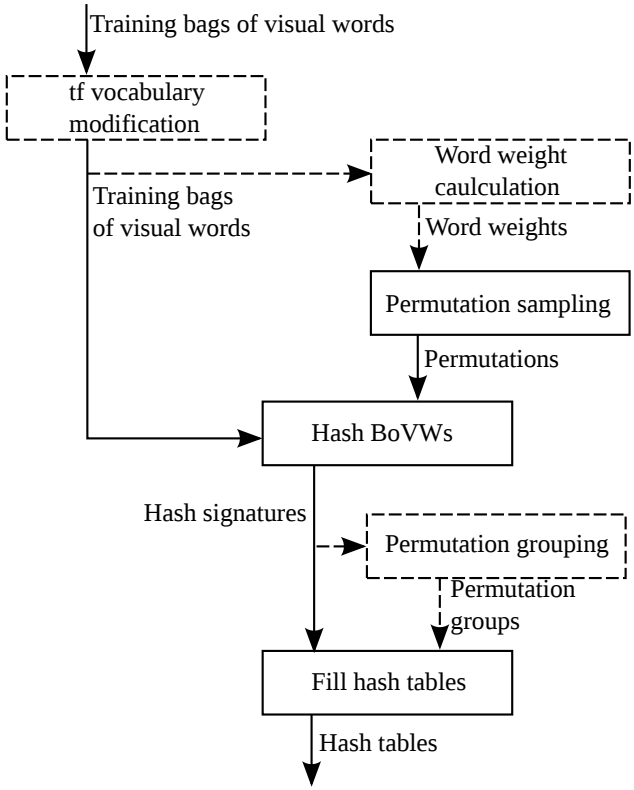


Figure 6.3: The hash table construction pipeline. Dashed rectangles and lines are optional extensions.

data, save the mapping and apply it to query BoVWs. In this case, we have to accept that either (a) unknown term frequencies cannot be incorporated without executing the offline pipeline from hereon again because the vocabulary size changes or that (b) the word-frequency pairs have to be reserved during construction time which causes a considerable blowup of the vocabulary.

A word weighting affects the sampling procedure of the MH permutation, as discussed in Section 4.1.2. The method would first estimate the weights given the training data and then pass them to the permutation sampling method. Without weights, we obtain uniformly sampled permutations, i.e. all words have the same weight. Both the word weight calculation and the sampling can be parallelized, but since both are done entirely offline and are relatively fast, neither needs to be sped up.

Permutation grouping is another optional extension that determines which permutations are grouped into the same sketch, cf. Section 4.1.4. This method has very high memory consumption and a long run-time, even if we do not consider all pairs of permutations. For every considered pair we need to model the whole joint distribution, which uses so much memory that we have to parallelize. Similar to the hash table construction, modeling joint distributions is not inherently parallel, because every document contributes in every joint. Parallelization involves reading all data multiple times for different joints and puts high load on the file server.

The permutation grouping is done after hashing, because we need hash values to estimate the probability distributions used for the grouping method. This is possible because changing sketch groups does not change results of MH functions.

Hashing can be done with both MH and GMH, with the restrictions discussed in Section 4.2.2: because term frequency weighting and permutation grouping become cumbersome and heuristic in the GMH context, we refrain from using them. If we use no permutation grouping, sketches are formed arbitrarily in the order they were generated.

6.1.3 Hash Table Queries

Querying of hash tables is only performed for test data, which means this part of the pipeline is only used during query-time. Techniques for carrying out the hash table queries are fairly straightforward. An overview is depicted in Figure 6.4.

The details of term frequency weighting, including query-time behavior, are already covered in Section 6.1.2. Hashing is also performed precisely as during hash table construction.

To query a hash table we simply look up the content of those cells in the hash tables that correspond to the query's hash signature. We intersect the sets of retrieved collisions within a sketch and join the collisions between different sketches, as explained in Section 4.1.1. Although this procedure is seems inherently parallel, every parallel query instance needs to load the hash tables which can become very large depending on the implementation details. Even if the memory of a cluster node is not exceeded, the start-up of several instances can put high load on the file server.

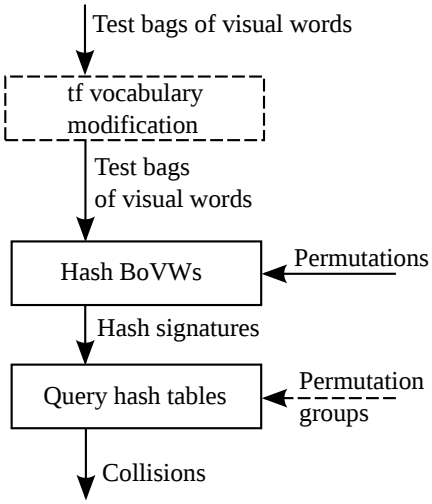


Figure 6.4: An overview of the hash table query pipeline. Permutations and permutation groups are generated during the hash table construction. Dashed components are optional extensions.

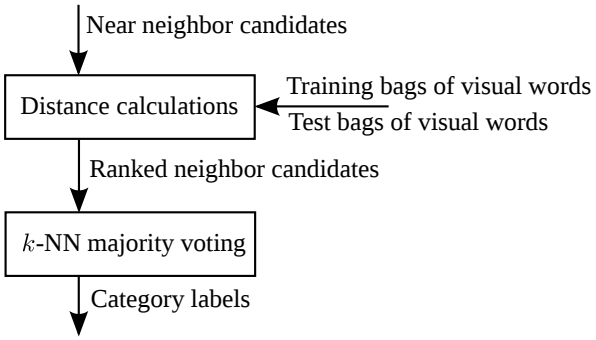


Figure 6.5: Overview of the final categorization using near neighbor candidates.

6.1.4 Ranking and Categorization

Only test data will be categorized, so this part of the pipeline is also only used during query-time. The procedure is very simple; its derivation and details are covered in Section 4.3. A schematic is presented in Figure 6.5.

Both the distance calculation and the k-NN categorization are inherently parallel. The main problem is that we need random access to individual BoVWs, as every query collides with a completely different set of training images.

6.2 Implementation Details

A careful planning of the pipeline is crucial in this work. The pipeline contains many small tasks that cannot be executed “in a row” on one image, because it needs informa-

tion from other images as well. Furthermore, most tasks work on all images, of which there are 1.4 million. Thus, every change of run-time of one step has a huge impact on the overall run-time.

To gain an impression of the magnitude of this impact, consider local feature extraction. Detecting affine interest regions and extracting SIFT regions with Mikolajczyk's software[1] takes 1.16 seconds for one particular image. Assuming all images take the same amount of time, the procedure would take almost 19 days. Now we use van de Sande's software[2] instead, extract scale invariant interest regions and rgSIFT descriptors, which takes 8.14 seconds on the same image. If we extrapolate this run-time to 1.4 million images, the procedure takes more than 131 days.

6.2.1 Infrastructure Analysis

The available infrastructure at the Max Planck Institute in Saarbrücken comprises of two machines with a large amount of RAM and a computing cluster.

- There are two machines with 500GB RAM and 48 and 32 cores respectively. Usually, interactive Matlab instances are run on those machines that use a large fraction of memory, so they are not fully available.
- Additionally, the MPI has a computing cluster where jobs are scheduled by the Sun Grid Engine (SGE). Depending on the required memory and run-time one user can run no more than 150–200 jobs at the same time, regardless of the current occupancy of the cluster; less if the cluster is very occupied.

  A job in the cluster can use up to 40GB RAM and run for up to 28 days. It is possible to use up to 16 threads per job, which does not affect the number of jobs running in parallel.

  Jobs in the SGE are shell scripts that are executed by the grid engine on a cluster node that has spare capacity. That means jobs are independent and cannot communicate by means provided by the SGE.

This means if we can parallelize a task and keep its memory usage below 40GB, the cluster provides a theoretical 150-fold speedup. A multithreaded program can utilize the two big machines better, yet if the memory consumption is below 40GB and the task can be parallelized in the cluster, the execution can be 50 times faster than on one of the large machines.

The ILSVRC2010 images are shipped as one directory per category, which contain all images of that category. On average, a directory contains about 1.000–1.200 images and it turns out that accessing such large directories is very slow. For example, listing the directory contents takes several seconds. To measure the impact of large directories we conducted the following experiment. First, we extracted 1.000 gzipped files, all in

[1] `http://www.robots.ox.ac.uk/~vgg/research/affine/`
[2] `http://koen.me/research/colordescriptors/`

the sample directory, to a local, temporary directory, then we extracted the same files from a gzipped tarball. Using individual files takes about 30 seconds, while extracting the tarball only takes one second.

By taking load off the file server we benefit in several ways. The obvious advantage is that a file access is finished earlier and the program runs faster. But when running a program 150 times in parallel in the cluster or even doing multithreaded file access in 2400 threads, the access time becomes much more crucial. If file accesses take too long, the file server suffer from under high load and processes requests at decreasing rates. So minimizing access effort of the file server is an important requirement of parallelization. There are two aspects of achieving this. We need a file and directory structure that causes minimal load and we also want to keep the number of file accesses to a minimum.

These considerations are contrary to our initial requirements of a fast development. Our intention is to concentrate on experimenting with ideas, so we are interested in readable and easy to debug file formats that can be handled with established tools and techniques.

6.2.2 Programming Language and File Formats

In this section, we discuss properties and peculiarities of Matlab and Python. We cover some pitfalls and details of the techniques and decide for a file format.

From the preceding analysis in Section 6.2.1 we can conclude the following requirements:

- We require a convenient high-level programming language that provides as many tools as possible to enable fast prototyping and development.

- The language needs a compact memory footprint, because we can use more cores if we stay below 40GB.

  To sketch out how important memory efficiency is consider GMH tables for $k = 5000$ sketches of size $s = 5$, i.e. we have 25.000 hash tables. Each table contains all 1.2 million training images. Disregarding all overhead and the actual hash table data structure, we can calculate the memory used only for storing the *content* of the hash tables' cells. If the training images are represented by 32-bit integers, the content requires $25.000 \cdot 1.2 \cdot 10^6 \cdot 4\text{byte} = 120\text{GB}$ of RAM. Conversely, assuming we have 500GB RAM, we can afford $(500\text{GB} - 120\text{GB})/25.000 = 15\text{MB}$ of RAM per hash table. In consequence, we depend on having a language without large overhead.

- The language should provide tools to work with large files without having to load them into memory entirely.

- The file format needs to condense data about many images into one file.

- The used file formats should not be programming language-specific.
- The file format should be examinable by standard tools.

The most popular choice, particularly in computer vision and signal processing, is Matlab because it has a large toolbox built in. Matlab ships many very optimized libraries and does implicit optimizations on large matrix operations. It started as a wrapping language for numerical FORTRAN algorithms and developed further with the focus of being in interactive "matrix laboratory". The powerful library provides means for programming on a high level of abstraction, hiding all details of complex operations.

Matlab is, however, problematic for large scale tasks. We require to save data concerning several images into one big file. The native Matlab solution would be a large, indexable structure such as a matrix or a cell array, which is saved into a MAT file. The problem with this approach is that it is impossible to access part of the saved data structure without loading the complete data into memory. This way the memory usage of a program is much higher, even if only considering one image at a time. Furthermore, debugging becomes more involving because it is impossible to inspect any aspect of a MAT file without loading it into memory.

Python is a modern programming language with many useful features and a broad standard library. There are ambitions to develop libraries for python that achieve the efficiency and comprehensiveness of Matlab's library. Numpy implements fast data types and manipulation operations in C, and Scipy extends Numpy to more advanced data structures and numerical algorithms. These libraries are not as broad as Matlab's, but allows Python to be similar to Matlab in terms of memory usage and data manipulation performance.

Most parts of the pipeline outlined in Section 6.1 do not require any complex algorithms that are missing in Python. Furthermore, both languages can be extended by compiled C code to speed up bottlenecks that are caused by slow execution of byte code. Python lacks proper multithreading support because of a global interpreter lock, which requires multiple processes to achieve real parallelism. Matlab does not support explicit threading either, only implicit threading of some library functions and "pools of workers", which are processes as well. The problem of multiprocessing is that shared memory is small and data serialization is slow. We found hyperthreading easiest to achieve by C kernels using openMP.

The initial MH implementation at the Mobile Multimedia Processing group was implemented in Matlab. The pipeline was used for image clustering and landmark discovery [WHL10], i.e. for finding collisions in one big set of images. It uses plain text files with a newline-oriented formatting to keep them human-readable. The pipeline produces one file per image, which was not an issue in the RWTH computing cluster. For the cluster in Saarbrücken, we need to combine groups of files into one new file to reduce the load of the file server. We decided to use gzipped tarballs to keep the file formats the same and to still be able to examine the new files in the command line.

Python has a comfortable way to iterate through tar files without loading the whole content into memory or extracting the content to a temporary directory. Unfortunately, Matlab is unable to read gzipped or tar files, the only way is to extract a tarball to a temporary directory, subsequently listing the directory's content and then reading each file.

Chapter 7

Experiments and Evaluation

In this chapter, we evaluate the methods introduced in Chapter 4 on the ILSVRC2010 dataset. First, we establish two baselines, guessing and Support Vector Machines (SVMs) on the features we will use later on. We perform exact k-NN categorization as a reference for experiments involving hashing. Then, we examine general effects of hashing parameters exceeding traditional ranges. Finally, we investigate on error rates which are obtained by hashing and k-NN classification.

Specifically we are interested in answers to the following questions:

- How does the vocabulary size affect the performance? Are there preferable vocabulary sizes for categorization?
- How do distance measures perform on sparse BoVWs? Is there a best choice? If so, does the preference transfer to the hash collision ranking?
- What is the effect of sketch sizes and the number of sketches? Is there an optimal choice?
- Does the quality of collisions correlate with the error rate?
- To what extent do the extensions improve the trade-off between effort and performance?

7.1 Baseline Setups

In this section, we establish two baselines to be able to assess the order of magnitude of later error rates. Guessing is method that should always be outperformed. SVMs are stable classifiers that are known to perform well, so we use their results as a goal and to assess the quality of our extracted features.

Table 7.1: Baseline error rates for the data base using the same labels for all queries.

Data set	Error measure	top 5	top 1
ILSVRC2010 val	flat	99.5%	99.9%
	hierarchical	8.1	12.5
tiny	flat	95%	99%
	hierarchical	8.1	13.7

7.1.1 Guessing

For a baseline comparison we analyze which error rates would be achieved by assigning the same category label to all test data. A categorization approach should of course be able outperform this kind of guessing, because it can base the label prediction on the specific query images.

The evaluation for the flat error is simple, because all categories occur with the same frequency in the test images, cf. Section 5.1.2. This means, the choice of category does not matter for this baseline. The ILSVRC2010 database contains 1.000 categories, so using only one label for all test images classifies 999 of the categories wrongly. For the top 5 error we are allowed to provide 5 labels. The best strategy is to use 5 different labels, but the particular choice does not matter as before. The other 995 categories are always misclassified and they all appear with the same frequency. For the tiny set, the calculation can be done analogously with the difference that there are only 100 categories. The results can be seen in Table 7.1.

The hierarchical error weights misclassifications depending on the semantic distance between the predicted label and the true category of an image. For the top 1 hierarchical error we simply use each category to classify all images and choose the category that minimizes the hierarchical error. Let $\mathcal{C}$ the set of all categories, $\mathcal{I}$ the test images and C^{hier} the hierarchical cost matrix. Then the best choice is to label all images in $\mathcal{I}$ with the category

$$k_{\text{top 1}} = \operatorname*{argmin}_{k \in \mathcal{C}} \sum_{I \in \mathcal{I}} C^{\text{hier}}_{c_I, l} \,. \tag{7.1}$$

For the evaluation set of ILSVRC2010 the best category is $k_{\text{top 1}} = 602$ (“shredder”) and has a hierarchical error of 12.5. In the tiny set it is $k_{\text{top 1}} = 744$ (“crossword puzzle”) with a hierarchical error of 13.7.

The top 5 hierarchical error allows 5 predictions for each image and uses that with least cost. To search through all possible predictions would be infeasible, because there are $\binom{1000}{5} > 8 \cdot 10^{12}$ different label tuples. Instead, we choose the best of some randomly generated tuples so the choice will not be optimal. The best tuple in a set of 10,000 random tuples was $(7, 148, 355, 668, 907)$ (“strawberry”, “male orchis”, “English setter”, “ambulance”, “swimming trunks”) with a hierarchical error of 8.1 on

Table 7.2: Categorization performance of linear SVMs on ImageNet's features.

	top 5		top 1	
Training images per class	flat	hier	flat	hier
100	80.03%	6.77	91.01%	10.35
700	70.82%	6.09	85.94%	9.57
all (ca. 1,200)	70.17%	6.08	85.33%	9.58

Table 7.3: Error rates of linear SVMs on our features. The minimum of each column is emphasized.

	top 5		top 1		
Vocabulary size	flat	hier	flat	hier	run-time
500	99.50%	12.11	99.90%	17.27	
1,000	74.16%	6.56	87.68%	9.78	
10,000	**70.35%**	5.91	84.35%	**9.25**	17h
50,000	70.57%	**5.88**	84.54%	9.33	22h
100,000	70.48%	5.92	**84.21%**	9.30	25h
1,000,000	73.94%	6.48	86.13%	9.78	53h

both the validation and the tiny set. For comparison, an arbitrary tuple of categories $(1,2,3,4,5)$ has a hierarchical error of 12.1.

7.1.2 Support Vector Machines

In this section, we report error rates of SVMs as a second baseline method. We also compare the performance of SVMs on our features with features published by ImageNet.

ImageNet provides a demo setup including features for all images and a categorization script that builds on top of liblinear[1]. The features are SIFT descriptors (cf. Section 3.3.1), sampled on a regular grid (cf. Section 3.1), clustered into a vocabulary of 1,000 VWs, and summarized in BoVWs (cf. Section 3.4). The categorization script trains linear 1-vs-all SVMs on 100 images per category and calculates error rates for the evaluation set.

Table 7.2 shows all error rates of the standard demo setup on the validation set. Training on 100 training images per class, it yields a top 5 flat error of 80% and a top 5 hierarchical error of 6.7. If we invest more time and memory and train on all images, the top 5 flat error drops to 70% and the top 5 hierarchical error drops to 6. The procedure takes about 10 hours, if we train on all 1,2 million images.

[1] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`

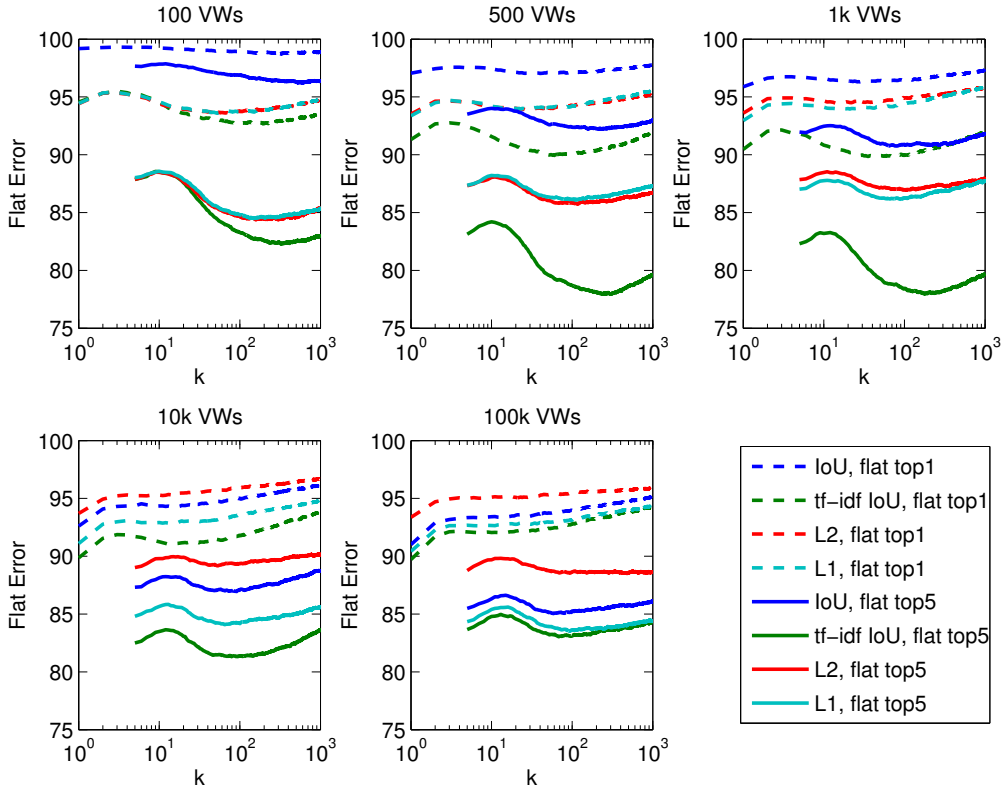


Figure 7.1: Flat error rate for k-NN categorization. Different plots show various vocabulary sizes, the x axis denotes the number of neighbors.

We extracted SIFT descriptors (cf. Section 3.3.1) of Hessian-Affine interest regions (cf. Section 3.2) and clustered them into vocabularies of different size between 500 and a million VWs and constructed BoVWs. Best results are obtained for vocabulary sizes between 10,000 and 100,000. The flat error rates are approximately the same as for the ImageNet features, while the hierarchical error is slightly better. The results are shown in Table 7.3 and are to be compared with the last row in Table 7.2.

7.2 Reference Setup: Exact k-Nearest Neighbor

In this section, we analyze the categorization performance of k-NN with different distance kernels, cf. Sections 4.3 and 3.4.2. The results serve as a comparison for the later experiments, as MH and GMH approximate similar distance measures. We will also compare the results to k-NN categorization on the tiny set.

The goal of this setup is to perform *exact* k-NN categorization in the sense that the nearest neighbor search will search all training images exhaustively. Later experiments will also categorize with k-NN, but do not exhaustively search all images.

In this experiment we calculate the distance between each pair of training and validation images, keep only the closest k training images per validation image, and perform a majority vote on the labels of the obtained k images. We employ four different distance kernels for the experiment, which were discussed in Section 3.4.2:

L1 and L2 denote the Manhattan and Euclidean distances respectively. We use these conventional distances for comparison, to assess the performance of the hashing related distances. For these distances, we normalize all BoVWs first, i.e. scale each BoVW $\mathbf{x}$ such that $\|\mathbf{x}\|_d = 1$ where d is 1 or 2 respectively. Then the distance functions are applied.

IoU and tf-idf IoU are also introduced and discussed in Section 3.4.2. In these cases the BoVWs are not normalized, yet we can think of the set assumption of the IoU similarity as a normalization that clips term frequencies that are greater than 1. One subtlety is that IoU and tf-idf IoU are both similarity measures, while k-NN requires distance functions. Actually, k-NN merely requires a function that allows ranking in which similar images receive a low score and dissimilar images receive high scores. This is achieved by using negative similarities, which yields not a distance measure but an appropriate ranking function.

Figure 7.1 shows the top 1 and top 5 flat error rate for all four distances for different vocabulary sizes. Consistently through all vocabulary sizes, the best error rate is achieved by the tf-idf weighted IoU similarity. The performance is best for vocabulary sizes of 500 and 1.000 and for roughly $k = 200$ neighbors, namely ca. 78%. For larger and smaller vocabularies, the performance of tf-idf IoU decreases.

The error rate curves display a sinus-like shape. This shape might be due to new neighbors that are found for $k = 5..12$ sometimes have a different label, because images from other classes have similar appearance. Until the number of neighbors with the desired label gain majority, the ranking of neighbor labels is arbitrary and is likely to reduce the performance. The frequency of images from the same category as the query among the new neighbors for k between 12 and 200 increases, so the majority voting performs better. Yet for higher values of k, the number of neighbors from the other categories increases faster than the number of neighbors from the right category, because we probably already retrieved all similar images from the correct category.

The hierarchical error exhibits similar properties as the flat error with the difference that the best hierarchical error of about 7 is obtained for roughly $k = 80$ and tf-idf IoU ranking, as shown in Figure 7.2. Furthermore, some curves monotonically increase instead of having a wave form. If the flat error improves while the hierarchical error increases, the number of misclassifications decreases but becomes worse in the sense of semantic distance.

If we compare the tendencies of distance measures through the various vocabulary sizes, there are several interesting characteristics. Let D be the number of dimensions, which is the size of the vocabulary.

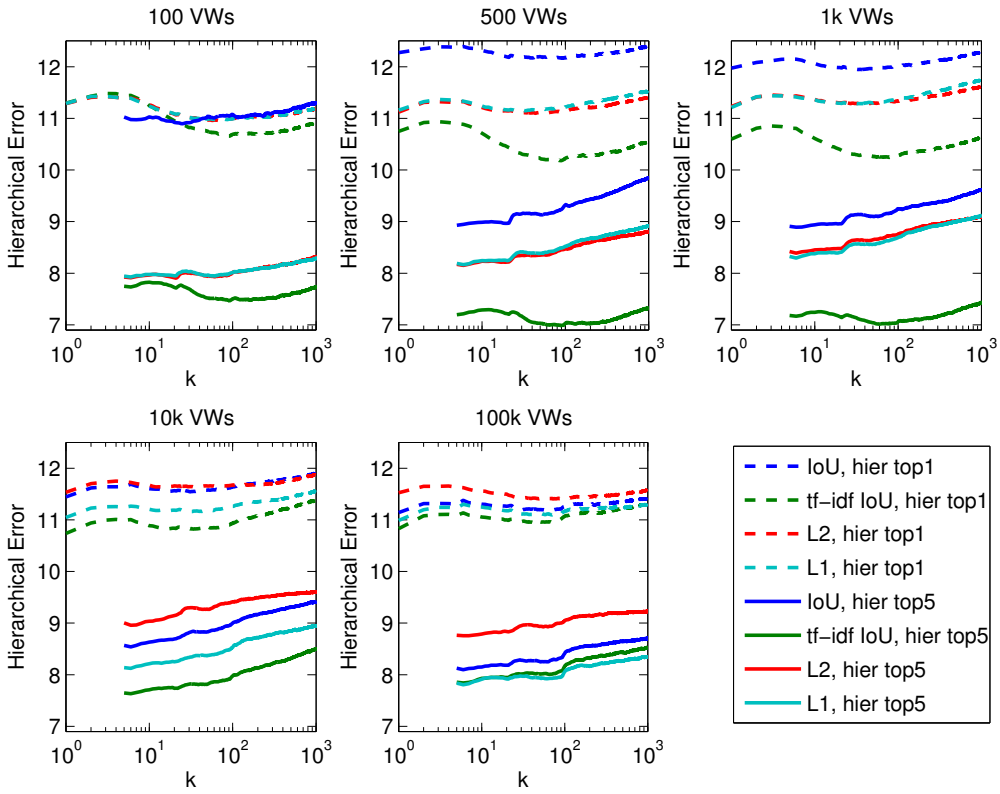


Figure 7.2: Hierarchical error rate for k-NN categorization. Different plots show various vocabulary sizes, the x axis denotes the number of neighbors.

Table 7.4: Runtime of exhaustive nearest neighbor searches. The time units are d for days and h for hours.

Vocabulary size	L2	L1	IoU similarity	tf-idf IoU similarity
100	4d, 13h	4d, 13h	4d, 5h	4d, 9h
500	4d, 19h	4d, 21h	5d, 12h	5d, 18h
1,000	8d, 23h	8d, 14h	11d, 11h	11d, 6h
10,000	23d, 2h	35d, 19h	39d, 5h	36d, 21h
100,000	890d, 0h	669d, 15h	611d, 22h	633d, 22h

- The L1 and L2 performances are similar for $D = 100$, but diverge as the dimensionality increases. The reason for this behavior is that the two distance measures converge for decreasing values of D. As discussed in Section 3.4.2, the L2 distance is not suitable for distances between histograms and this effect becomes more pronounced for large vocabularies.

- The IoU and tf-idf IoU performances become more similar for large D. As the number of VWs increases, words tend to occur at most once per image and so the term frequency becomes binary like in the set assumption of IoU. Furthermore, VWs appear in less of the images, because the VWs become more specific. So the document frequency approaches one and the inverse document frequency scores become uniform.

- The IoU performs particularly bad for small D. As D decreases, VWs tend to occur more than once and most information is contained in *how often* a word appears in an image and not in *whether* it appears. The IoU similarity discards this information due to the set assumption.

Table 7.4 shows the run-time for calculating the distance between all pairs of training and validation images including the required file IO. None of the kernels is consistently faster or slower than the others. The overhead for file IO appears to be roughly 4 days, although it increases as we split the workload to more jobs in the cluster to stay within the memory bounds.

Figure 7.3 displays the performance of k-NN with IoU and tf-idf IoU similarity for the tiny set. We can see similar trends as for the validation set: both similarities have different performance for small vocabularies and converge for large vocabularies.

The main purpose of Figure 7.3 is to investigate the effect of working on the tiny set instead of the standard ILSVRC2010 training and validation data. The best flat error of approximately 60% is obtained for k between 40 and 150 using 1,000 VWs. The best hierarchical error of 6 is obtained for the same parameter range. Compared to the optimum on the validation set, the flat error improves by 18 percent points, which is an improvement 23%, while the hierarchical error drops from 7 to 6, which is an improvement of about 17%.

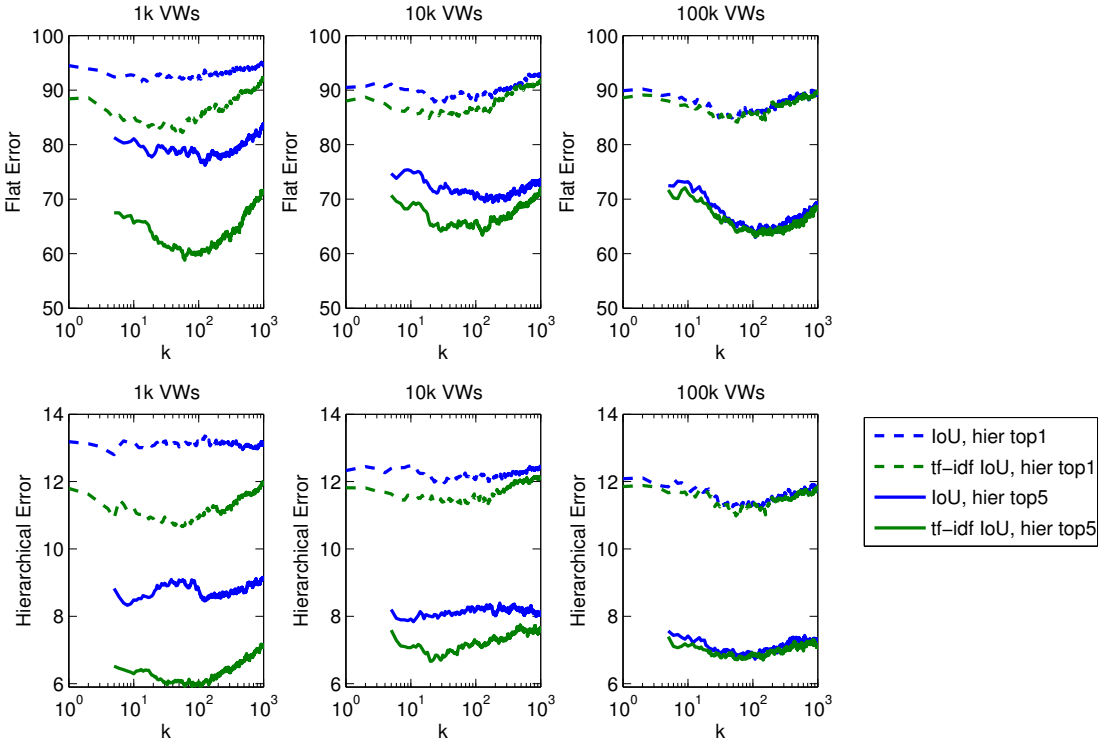


Figure 7.3: Flat and hierarchical error rate for k-NN categorization on the tiny set. Columns are different vocabulary sizes.

7.3 Effect of Hashing Parameters

This section evaluates the impact of the number of sketches k and the sketch size s on the performance characteristics of MH. The two main interests are the number of collisions per query, called *collision rate*, and the fraction to which the retrieved images actually are neighbors, loosely referred to as *approximation quality*. We deliberately relinquish the class information to investigate the approximation MH implies.

Standard MH is designed and employed for near-duplicate detection and as such, its typical parameter values cause a collision rate much lower than one collision per query. For categorization, the main concern is to increase the collision rate to a level where categorization is possible, while obtaining a maximal quality of retrieved neighbors.

Traditional parameter ranges for near duplicate detection are sketch sizes up to 3 and about 100 sketches [CPZ08]. We will investigate further than the typical boundaries to see trends. The goal is to obtain enough near neighbors among the top ranked images to achieve similar categorization results to exhaustive search.

7.3.1 Collision Rate

We do not know which collision rate is optimal for good performance, but we can look for sensible bounds. In Section 7.2, we obtained best results for about 100 neighbors, so we want at least as many collisions. Considering that we will encounter false positives, we certainly want more collisions. On the other hand we want our method to offer a speed advantage by retrieving a small set of neighbor candidates.

Figure 7.4 shows the collision rates as a surface over a grid of values for s and k, where each plot presents one vocabulary size. There are some holes in the plots for small vocabularies, because the number of collisions grows intractable. For one million VWs, there are holes for $s = 5$ and for $s = 4$ with small values of k, because the number of collisions drops to zero.

The collision rate decreases exponentially for increasing values of s. This trend shows linear in the figure, because of the logarithmic z axis. The impact of s increases for large vocabularies. For increasing k the collision rate increases linearly, as shown in Figure 7.5.

We can explain the effect of the vocabulary size with probabilistic thresholding (cf. Section 4.1.1) and the set assumption of MH. For small vocabularies, VWs have a high probability of appearing multiple times per image, so the highest variance is in *how often* a VW appears instead of *whether* it appears at all. If this term frequency is capped to 1, images become more similar, more image pairs are more similar than the threshold, and the number of collisions increases. If we increase the vocabulary size, the vocabulary samples the SIFT feature space more densely with VWs, i.e. VWs tend to be "split" into different VWs. This process tends to remove elements from the intersection of BoVWs and thus makes images more dissimilar.

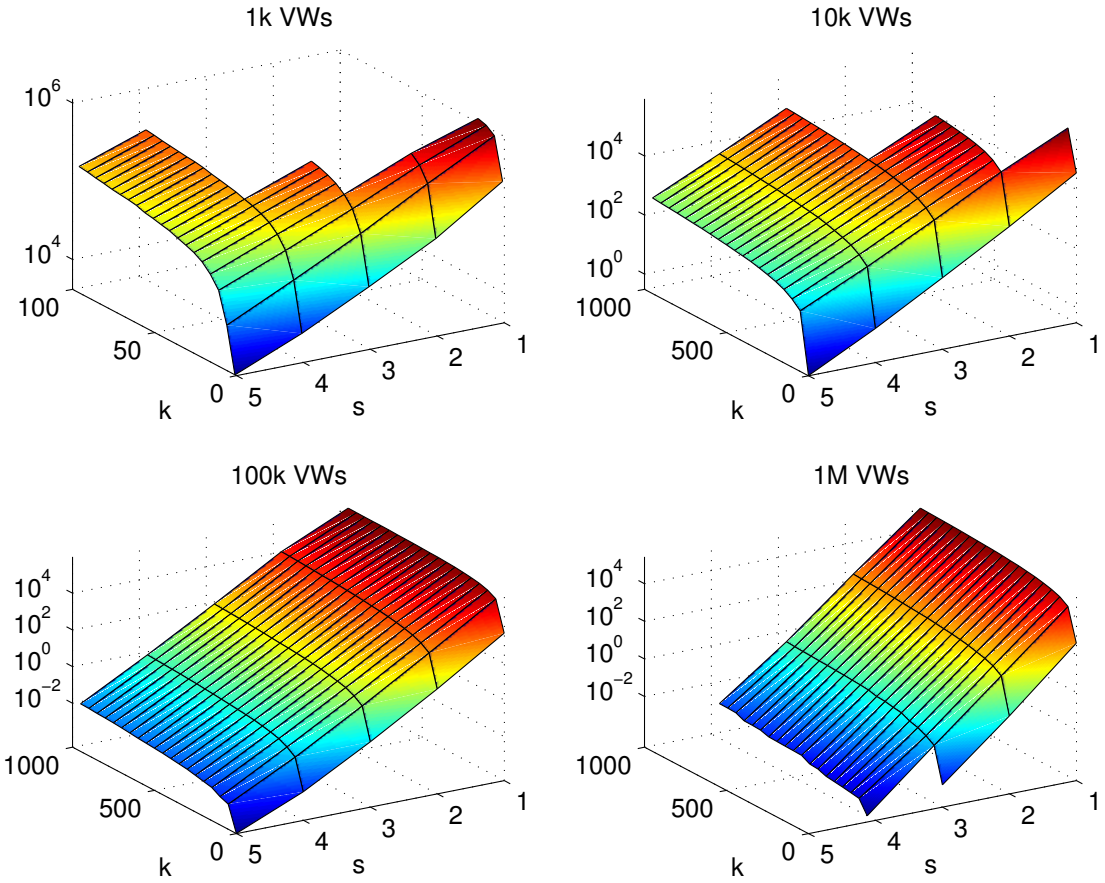


Figure 7.4: Collision rate of MH for various vocabulary sizes, number of sketches, and sketch sizes. The z axis is in log-scale.

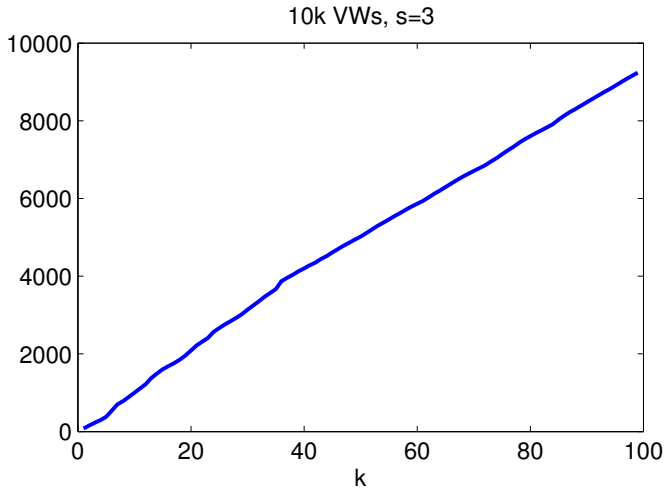


Figure 7.5: Collision rate over the number of sketches k for vocabulary size 10,000 and sketch size $s = 3$. For settings with more collisions, saturation is to be expected.

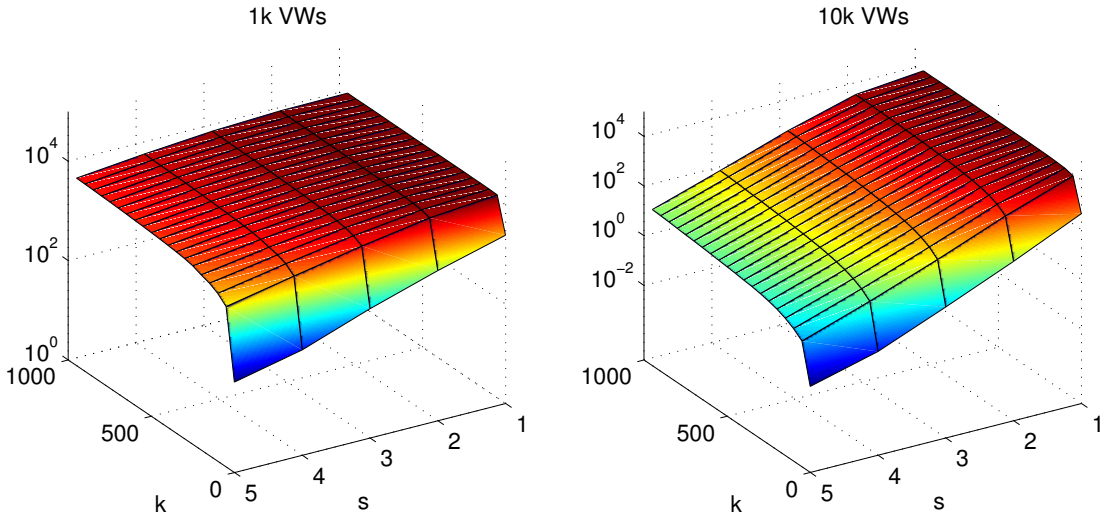


Figure 7.6: Collision rate of MH on the tiny set for various vocabulary sizes, number of sketches, ans sketch sizes. The z axis is in log-scale.

Collision rates for 1,000 and 10,000 VWs on the tiny set are shown in Figure 7.6. We see a similar behavior as for the validation set, but approximately two orders of magnitude less collisions. As there are no holes in the surfaces, we can see how the collision rate converges to the size of the training set.

7.3.2 Approximation Quality

In this section, we investigate how many of the nearest neighbors are actually retrieved. A perfect parameter setting would retrieve the 100 nearest neighbors while minimizing the number of additionally retrieved images. In practice, we encounter a lot of false positives, so there is a trade-off between the number of retrieved nearest neighbors and the total number of collisions.

For this experiment, we investigate the collisions of the validation set. We consider the 1,000 nearest neighbors of each query to be "good" collisions and investigate the quality of collisions in terms of the absolute and relative number of good collisions. Figure 7.7 shows the average number of good collisions per query and the average fraction of good collisions among all collisions per query.

The absolute number of good collisions consistently increases for increasing values of k, and more rapidly for small values of k. The curves flatten out for different values of k, depending on the vocabulary size. For 1,000 VWs the curve flattens out for approximately $k = 40$, while for 100,000 VWs it does not happen earlier than $k = 400$. Increasing the sketch size s causes the number of good collisions to decrease approximately exponentially, although this effect is less pronounced for 1,000 VWs. Note that these trends are similar to the collision rate in Section 7.3.1.

The relative number of good collisions improves for increasing values of both s

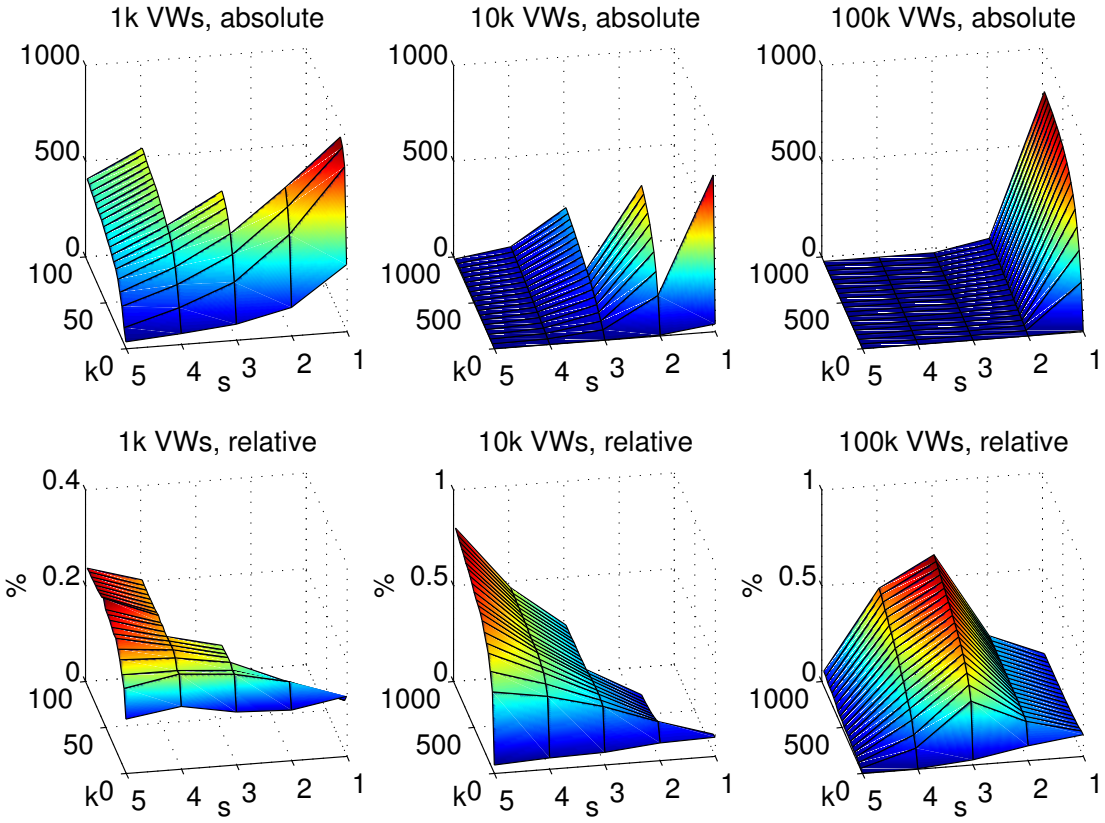


Figure 7.7: Collision quality for various vocabulary sizes. Top row shows the absolute number of nearest 1,000 neighbors retrieved on average. Bottom row show the fraction of retrieved images that actually are nearest neighbors.

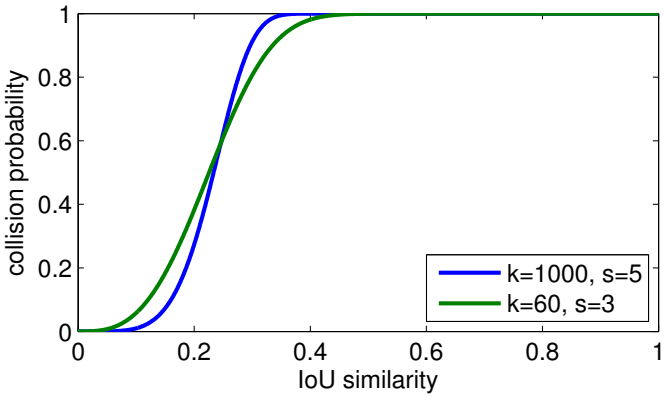


Figure 7.8: Collision probability over similarity for parameter settings $k = 1000, s = 5$ and $k = 60, s = 3$. Both parameter setting yield the steepest slope for a similarity of approximately 0.25.

and k with a curious exception for 100,000 VWs, for an explanation see below. For the vocabulary size of 100,000, more sketches k still improve the collision quality while larger sketches than $s = 3$ only decrease it.

Unfortunately, all relative frequencies of good collisions are rather low, namely below 1%. However, the number of good collisions we expect purely by chance when randomly selecting any training image is roughly $1,000/1,200,000 \approx 0.083\%$. This number is visible at certain points in the plots. For a vocabulary of size 1,000 and sketch size $s = 1$ almost all queries collide with almost all of the training images even for only a few sketches. By retrieving all training images we also retrieve the nearest neighbors and eventually the relative number of good collisions will converge to 0.83%. For 10,000 VWs we see similar behavior for larger values of k. We see similar relative numbers of good collisions for sketch size 1 and 5 for 100,000 VWs. Most other parameter settings perform better than chance.

We suspect that the decrease in collision quality for sketches larger than $s = 3$ for 100,000 VWs is caused by many queries that do not collide at all. While few collisions for a query can still contain a reasonable number of good collisions, zero collisions always contribute 0%. An insight into the number of queries that do not collide can be gained from the collision rate in Figure 7.4. For $s = 5$ and $k = 1,000$ we encounter a collision rate of approximately 0.01. This means there are at most $0.01 \cdot 50,000 = 500$ queries that collide or conversely at least 99% of the queries do not collide.

7.3.2.1 Probabilistic Thresholding Revisited

Theoretically, there are different settings for k and s that yield a similar similarity threshold (cf. Section 4.1.1) and a similar number of collisions. This reasoning leads to the idea that it is possible to obtain similar results with less and smaller sketches and thus less effort. To investigate such a case consider the statistics for a vocabulary size of 10,000 and the two parameter settings $k_1 = 1000, s_1 = 5$ and $k_2 = 60, s_2 = 3$. Using the

rule of thumb for the similarity threshold $\theta = (1/k)^{1/s}$, we obtain the thresholds $\theta_1 = 0.25$ and $\theta_2 = 0.26$ respectively for the two parameter settings. Although the thresholds are rather similar, the first parameter settings causes 551 collisions on average, while the second yields 5864, an order of magnitude more collisions. The average number of good collisions are 13 and 33 respectively, i.e. the first parameter setting is more efficient in terms of good collisions per collision.

The intuitive explanation for this behavior is that larger sketches are generally more specific even for a large number of sketches. A more principled insight is provided by the relation between similarity and collision probability. As discussed in Section 4.1.1, the collision probability of two images depends on their similarity, where the number of sketches and sketch size parameterize the relationship: $p_c = 1 - (1 - \text{sim}^s)^k$. Figure 7.8 visualizes the relationship for the two parameter settings from the preceding example. The parameters $k = 1000, s = 5$ reveal a steeper change from low to high collision probability. Compared to that behavior, the parameters $k = 60, s = 3$ trade more probability of colliding with similarity less than 0.25 for less probability of colliding with slightly more similarity.

7.3.2.2 Suitable Parameters

We aim for an efficient neighbor retrieval system that is able to find enough close neighbors while avoiding retrieval of dissimilar images. With the preceding results, we can attempt a first assessment of the parameter space.

Generally, increasing the number of sketches and sketch size improves the quality of retrieved images. Yet, increasing the sketch size decreases the number of retrieved images exponentially, which is hard to compensate with the number of sketches which has a much weaker impact. This means we can increase s for a higher quality and simultaneously drastically decrease the number of retrieved images. More sketches increase the quality further and the number of retrieved images, but also increases memory usage and run-time.

We can expect the best effort-result trade-off for a parameter setting that maximizes the collision quality while yielding enough near neighbors, e.g. 100 neighbors. So a good strategy seems to be using as many sketches as possible without exceeding the resources and then increasing the sketch size until the number of collisions drops too low, i.e. the categorization performance decreases due to missing collisions. However, the criterion of the nearest one thousand neighbors being "good" collisions is weakly motivated and its impact on the k-NN categorization performance is unclear.

7.3.3 Frequency Analysis of Features

This section examines the expected impact of the vocabulary size on GMH. Standard GMH can only place sketch regions on VWs that are unique to an image, as discussed in Section 4.2. Thus, a high probability of VWs occurring multiple times in an image has a direct impact on the GMH performance.

Table 7.5: Frequency of VWs per image for various vocabulary sizes. "# IR" denotes the total number of interest regions, counting multiple occurrences of the same word multiple times. "# unique VWs" and "# not unique VWs" count every word at most once, so the two rightmost columns do not add up to the second column.

Vocabulary size	# IR	# unique VWs abs. / rel.	# not unique VWs
100	954.4	9.6 / 1%	75.9
500	954.4	95.8 / 10%	182.6
1,000	954.4	188.5 / 20%	210.2
10,000	954.4	594.9 / 62%	148.1
100,000	954.4	812.4 / 85%	65.1
1,000,000	954.4	881.4 / 92%	34.5

The experiments with exact k-NN categorization (cf. Section 7.2) showed most potential for small vocabularies. Intuitively, we want a less specific vocabulary than for specific image retrieval, so the process works on rough shapes instead of precise appearances.

Table 7.5 shows statistics of how many VWs are unique. Images contain on average 954.4 interest regions. When their SIFT features are quantized into VWs, some are mapped to the same VWs while other VWs will be unique. The column "# unique VWs" displays the average number of VWs that appear exactly once per image and their relative number compared to the number of interest regions. The column "# not unique VWs" shows the average number of VWs that appear multiple times per image. A VWs that appears n times in an images is only counted once in the latter column, so the columns do not add up to the number of interest regions.

For a vocabulary of size 1,000, GMH would consider at most 20% of the features in an image as a sketch region, and less if features happen to have too few features in their scale-space neighborhood (cf. Section 4.2). The decision of whether a VW is a candidate for a sketch region or not is done per image. So a word w can be a primary hash value in an image I_1 if it is unique in I_1, but if it appears for example twice in I_2 it will never be a primary hash value of I_2. This means if the primary hash value of I_1 is w w.r.t. to a certain sketch, I_1 and I_2 are unable to collide on this sketch, even though both images contain the VW w.

Effectively, similar images are less likely to collide, because the set of their common VWs is artificially reduced. These results motivate the multi-region GMH approaches introduced in Section 4.2.3.

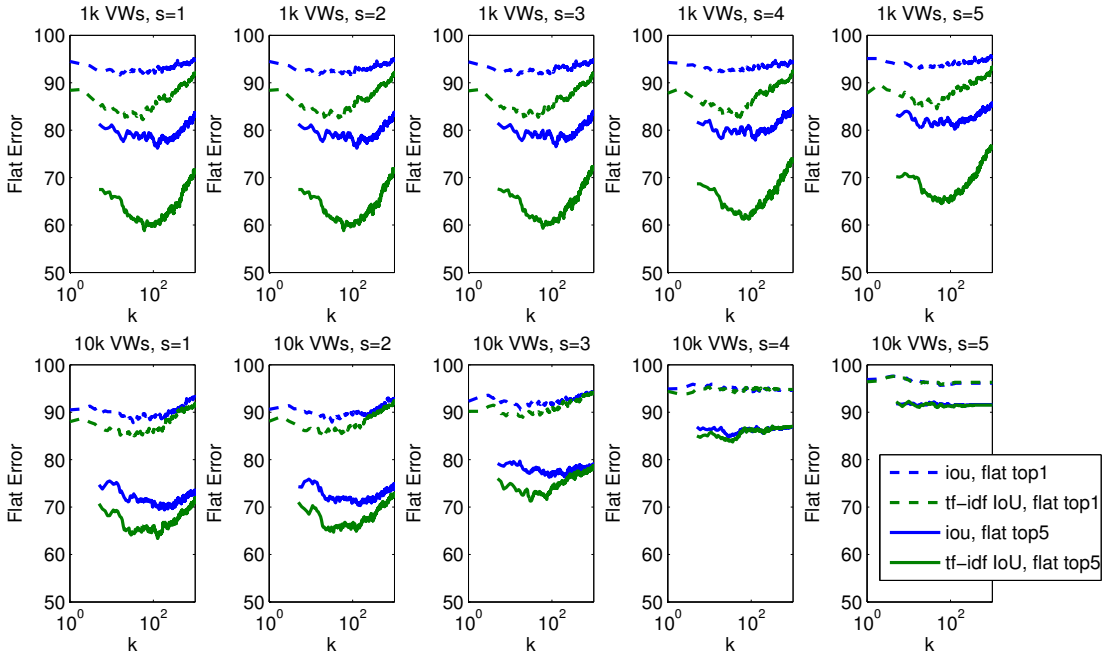


Figure 7.9: Flat error rates of k-NN categorization on the MH collisions for various sketch and vocabulary sizes. All experiments used 1,000 sketches. k denotes the number of used neighbors, different colors denote different similarity measures for re-ranking.

7.4 Min Hash

In this section, we present error rates for k-NN categorization on MH collisions. We investigate the trade-off between the number of collisions and the categorization performance and the effect of various extensions. All experiments are done on the tiny set.

Figure 7.9 shows the flat error rate for 1,000 and 10,000 VWs, all sketch sizes from 1 to 5, and 1,000 sketches. The dashed lines show the top 1 flat error and the solid lines the top 5 flat error. We omit hierarchical error rates, as they reveal similar behavior. The blue lines show the flat error for using an IoU kernel for k-NN classification, while the green line shows the error for a tf-idf weighted IoU kernel. The x axis shows the number of neighbors for k-NN categorization.

For 1,000 VWs and sketch size 1, we obtain a collision rate of approximately 12,000, i.e. almost all images collide with the entire database. This way, the plot looks the same as for k-NN with exhaustive search, cf. Figure 7.3. For larger sketch sizes, the flat error gradually increases as the collision rate decreases. Note that the tf-idf IoU similarity re-ranking yields strictly better results than the standard IoU similarity, even though MH approximates the IoU similarity. The optimal number of nearest neighbors is usually close to 100.

To put these results into the context of their collision rate, we turn each of the

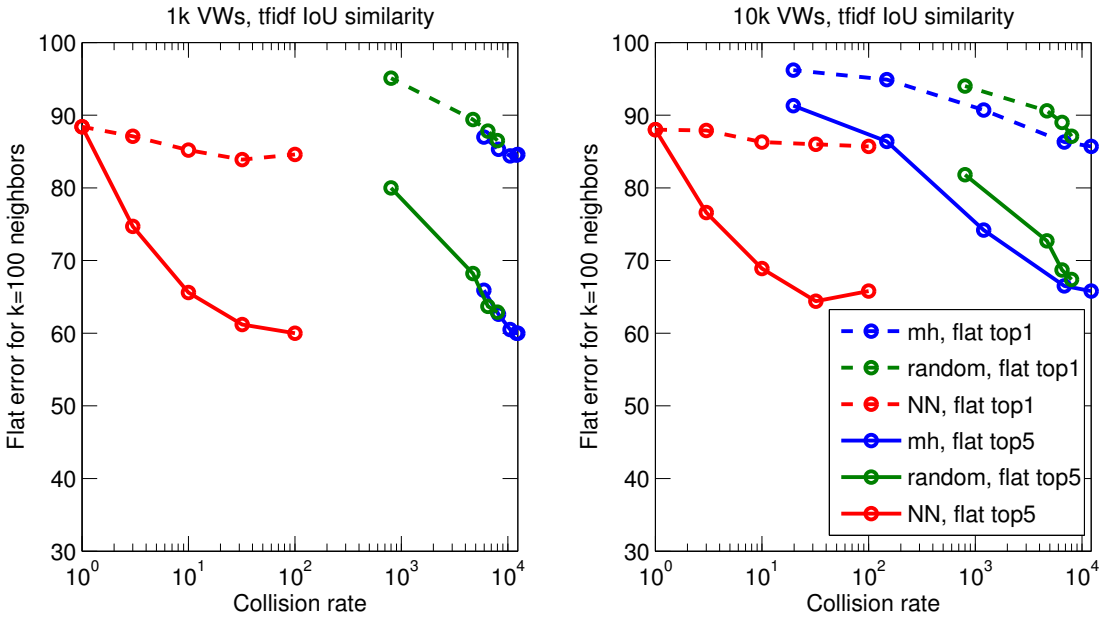


Figure 7.10: Error rate of 100-NN categorization with a tf-idf IoU kernel on MH collisions.

plots in Figure 7.9 into a point in an error rate over collision rate plot. This is only possible by representing the performance, which is the development of the error rate over varying number of neighbors, with merely one error rate. We choose the error rate for 100 neighbors and a tf-idf IoU kernel, because it proved to be close to the minimum of the preceding error rate plots.

Figure 7.10 summarizes the plots of Figure 7.9 in the previously explained way. The x axis shows the collision rate in logarithmic scale from an average of one collision to about 10,000 collisions, i.e. almost all queries collide with the entire database. We desire a low error rate with few collisions, so we want to move the error points to the lower left. The red points show the result of an optimal hash function that always retrieves the closes images first. A hashing method that approximates the tf-idf IoU similarity cannot outperform those points.

The blue dots visualize the error rate of k-NN categorization on MH collisions over the average number of retrieved images. In the plot for 10,000 VWs, we see sketch size $s = 1$ as the rightmost point. The next blue point to the left represents $s = 2$ and so on. We see that the reduction of collision rate causes the error to increase. All methods that cause queries to collide with the entire database for the least restrictive parameter setting will converge to the point with about 10,000 collisions and an error rate of roughly 66%. The shape of this curve visualizes the trade-off between speed and error we can achieve with MH. We strive to move the curve down or to the left with extensions and GMH. It is, however, important to understand that the connecting lines between points are for the ease of visual recognition of corresponding points. There is no simple method to obtain the collision rate–error points lying on the line.

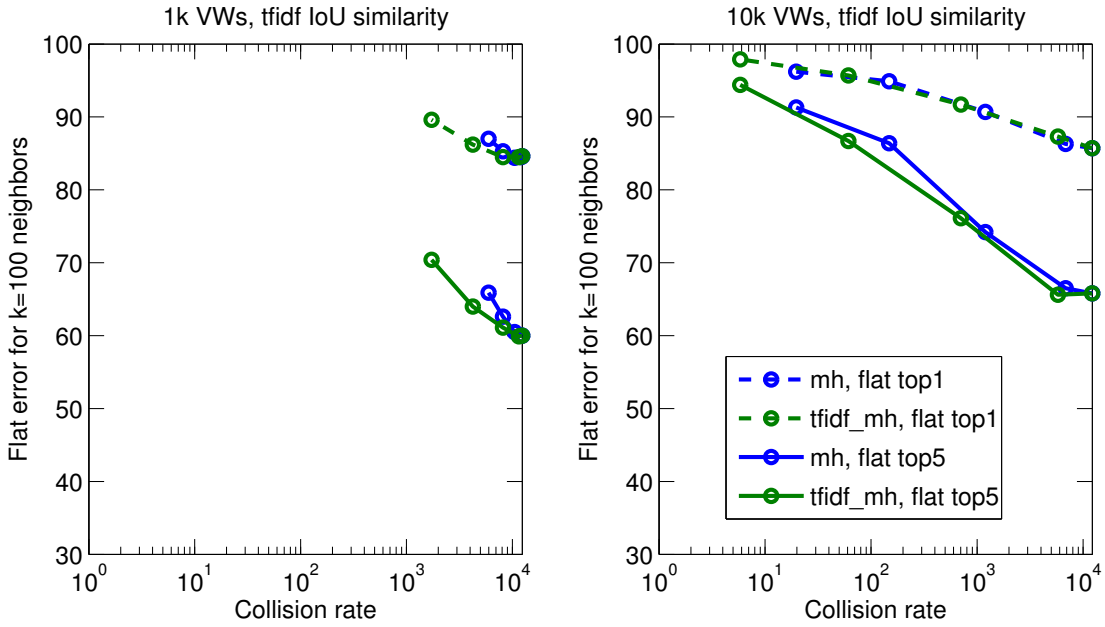


Figure 7.11: Error rate of 100-NN categorization with a tf-idf IoU kernel on MH and its tf-idf MH collisions.

The green "random" curve in Figure 7.10 is created using random training images for neighbor candidates during k-NN categorization instead of hash collisions. In detail, we first fix a desired number of collisions per query and then we draw random images per query. This means the number of collisions is the same for all queries, whereas for hashing, every query collides with a different amount of training images.

For 1,000 VWs, the random and MH curves are very close, while this is not the case for 10,000 VWs. As discussed before, the set assumption of the IoU similarity causes images to be more similar for small vocabularies, so we see much higher collision rates, even for restrictive settings of s. Furthermore the omission of term frequencies degrades its ranking quality. We suspect that the IoU similarity is inapt of ranking those images high that are beneficial to classification.

7.4.1 tf-idf Min Hash

Figure 7.11 compares the categorization error rates on standard MH collisions and the collisions of the tf-idf MH extension. We see that tf-idf MH consistently reduces the collision rate without increasing the error rate as rapidly as standard MH. Overall, the trade-off curve of tf-idf MH is better than for standard MH, especially for small vocabularies.

The reasons for these improvements are covered in Sections 4.1.2 and 4.1.3. In short, we bypass MH's set assumption and weight words. The former is essential for good performance with small vocabularies, as words appear more frequently. The latter is a general method to increase the influence of seldom words, as they are more

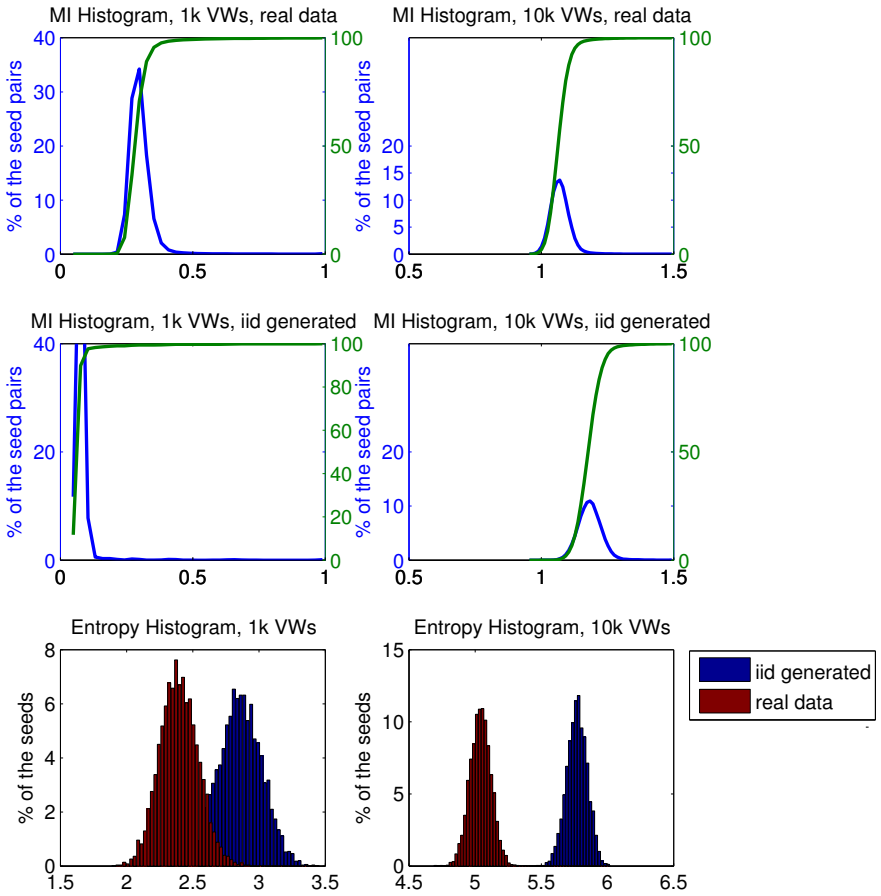


Figure 7.12: Histogramms of entropy and mutual information of real and generated data. The first two rows show the distribution of mutual information and its cumulative distribution, so the blue and the green y-axis are both in percent of seed pairs.

distinctive and allow more specific collisions.

7.4.2 Permutation Grouping

This section evaluates the permutation grouping extension to MH, as discussed in Section 4.1.4. We compare the properties of the mutual information distribution with [BCI08] and then investigate the impact on the results.

For this experiment, we hash all images and build a distribution of hash values for each permutation, which defines the hash function. The distributions are used to estimate the entropy of a permutation and mutual information of pairs of permutations. For further details, please refer to Section 4.1.4.

The last row of Figure 7.12 shows histograms of entropies of real and i.i.d. generated data for 1,000 and 10,000 VWs. We can verify a lower entropy for real data than for generated data. The first row shows a histogram of mutual information of real data for both vocabulary sizes. Compared to the histograms of generated data in the

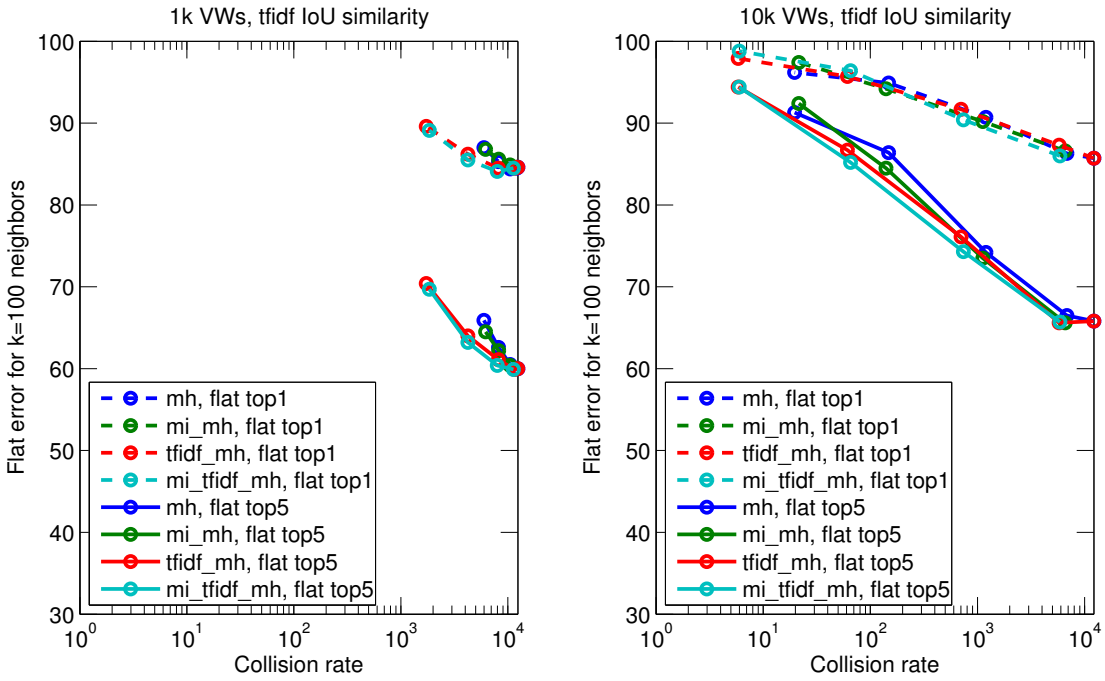


Figure 7.13: Error rate of 100-NN categorization with tf-idf IoU kernel on collisions of the permutation grouping extension. The prefix "mi" denotes the permutation grouping extension.

second row, we do not see the large tail in the distribution that is usually caused by "unfortunate" combinations of permutations, which was reported by [BCI08].

Reasons for the missing tail can lie in the distribution of the BoVW data or in the fact that we do not calculate mutual information of all possible pairs. Due to the size of the joint distribution and the large number of sketches, we do not calculate the mutual information of all 25,000,000 pairs. Instead, we form groups of 100 hash functions and only consider pairs within these groups, so effectively we calculate only mutual information for 500,000 pairs.

Figure 7.13 shows the effect of the permutation grouping extension on both standard MH and tf-idf MH. With the exception of sketch size $s = 5$, we see a consistent but small improvement of error rate. Note that the original intention of permutation grouping is the reduction of the number of collisions.

The improvement of the error rate instead of the collision rate might be due to better collisions. If better groups of hash functions are more specific and yield less false positives, this could decrease the error rate.

One of the advantages of permutation grouping is that it is possible to generate more hash functions than we need and have a principled way to omit those which improve the sketches least. We used a set of 5.000 hash functions and since the sketch grouping process uses every hash function only once, we have to use all hash functions for sketch size $s = 5$. Thus, none of the hash functions are discarded for $s = 5$, which

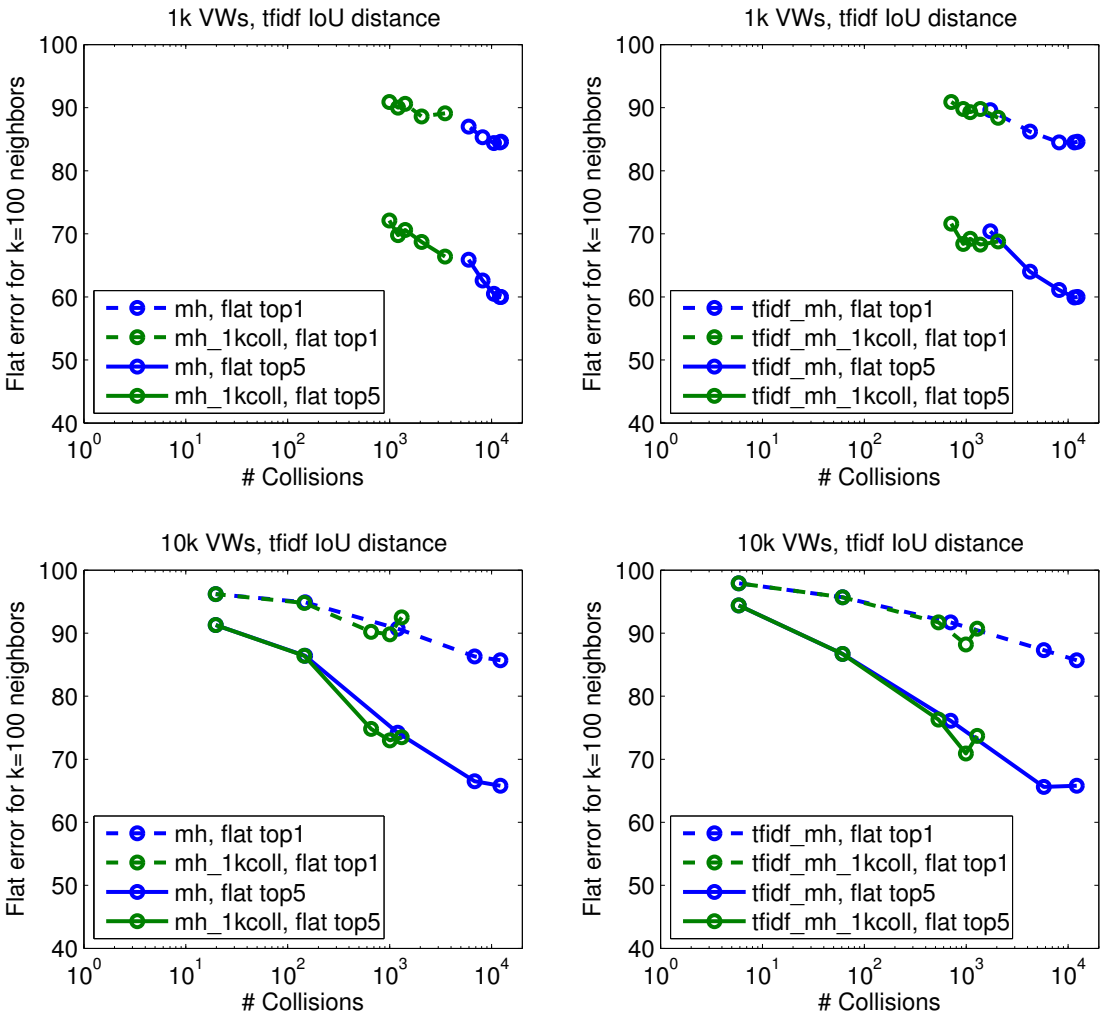


Figure 7.14: Error rate of 100-NN categorization with tf-idf IoU kernel on collisions of the threshold adaption extension. The left column shows the effect on MH, the right column the effect on tf-idf MH.

can lead to unfortunate last choices during sketch grouping. We suspect that this is the reason for worse performance for $s = 5$.

7.4.3 Threshold Adaption

In this section, we investigate threshold adaption as presented in Section 4.1.5. For the experiment we stop querying new sketches after we already retrieved 1,000 collisions.

Figure 7.14 shows the effect of the threshold adaption extension. Both colors correspond to the same method with the difference that the green curve uses threshold adaption. This means, we move the rightmost blue point to the corresponding rightmost green point by omitting all but roughly the first 1,000 collisions. We do not obtain

exactly 1,000 collisions for all queries for two reasons:

- The collisions of the last used sketch, in general, overshoots our threshold. We have no reason to prefer one of the collisions of the last sketch over another, so we keep all of them. The amount of collisions that exceed the threshold is usually small, but it increases for small sketches because small sketches produce more collisions.

- Larger sketches are more likely to yield less than 1,000 collisions for a query. As the sketch size increases there are more queries with less than 1,000 collisions and the number of collisions for those examples reduces further.

Thus, the number of collisions for a single query is rarely exactly 1,000, which consequently also applies for the mean of collisions. Moreover, we can observe that the ordering of sketch sizes from 5 to 1 is still present after applying threshold adaption for exactly the above reasons.

For the vocabulary of size 10,000, we see that the points for sketch size 1 to 3 move to approximately 1,000 collisions. Unfortunately, the process of reducing the number of collision also increases the error rate. The resulting error rate is approximately similar to the error rate we obtain by increasing the sketch size. For 1,000 VWs we see a similar trend, but the results have less collisions.

The results suggest to employ threshold adaption if we want to make sure we do not exceed a certain number of collisions. With this method we can achieve collision/error rate points on the connecting line between points. Yet the method seems to be quite unstable, as we encounter a difference of about 5 percent points of error rate for tf-idf MH and 10,000 VWs between sketch sizes 1 to 3 (cf. bottom right plot in Figure 7.14).

7.5 Geometric Min-Hash

In this section, we present error rates for k-NN categorization on GMH collisions. We analyze how different extensions affect the trade-off between the collision rate and the categorization performance. All experiments are performed on the tiny set.

Figure 7.15 shows a comparison between standard MH and GMH. In contrast to MH, GMH is plotted for sketch size 2 to 5, because the geometric modification is pointless without secondary hash functions, cf. Section 4.2. For 1,000 VWs, we see a significantly different behavior between MH and GMH for sketch sizes 4 and 5. The collision rate drops considerably and the error rate increases. Compared to this behavior, the results for MH and GMH on the larger vocabulary are rather similar.

The reason that GMH is inapt to outperform MH are probably the small vocabulary sizes. As discussed in Section 7.3.3, GMH only uses 20% of the interest regions for 1,000 VWs and only 62% for 10,000 VWs.

The results of multi-region extensions presented in Section 4.2.3 and 4.2.4 are shown in Figure 7.16. We see that all modifications outperform standard GMH in

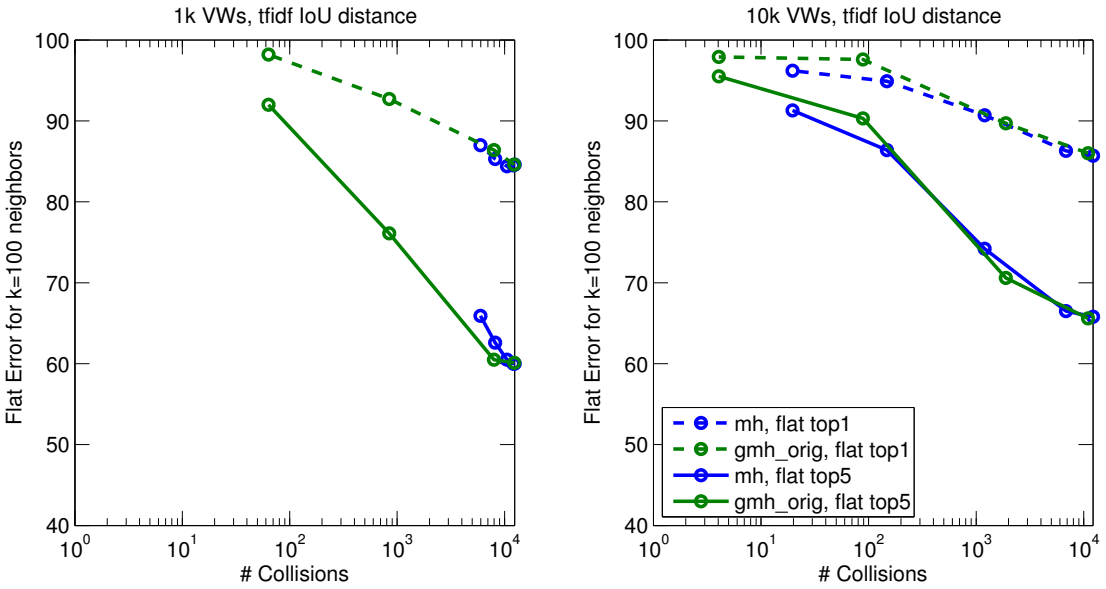


Figure 7.15: Error rate of 100-NN categorization with tf-idf IoU kernel on GMH collisions. "gmh_orig" denotes the standard GMH procedure.

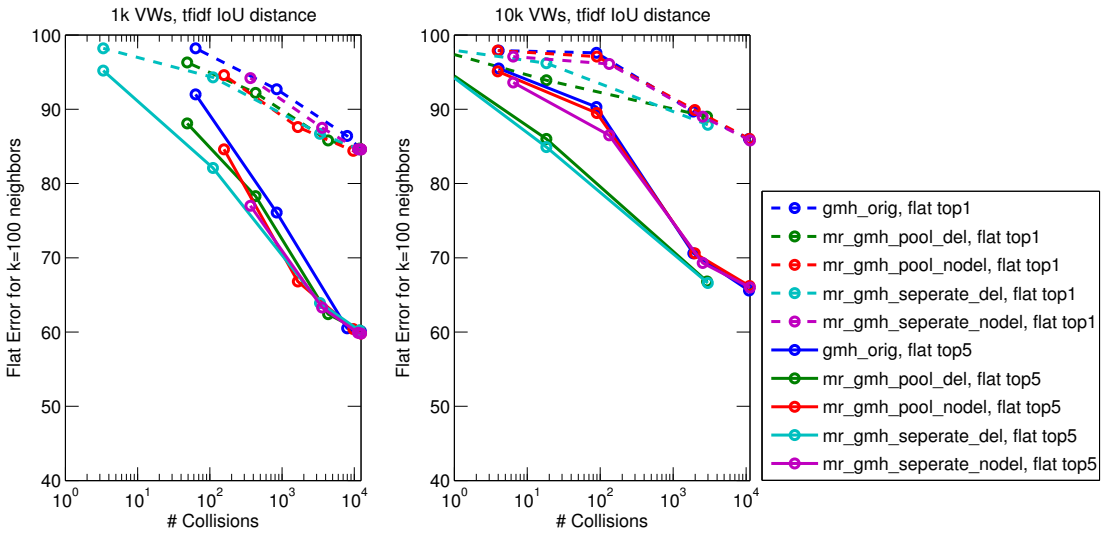


Figure 7.16: Error rate of 100-NN categorization with tf-idf IoU kernel on GMH collisions. "pool" denotes the merging of all corresponding neighborhood regions, "separate" denotes inserting all corresponding regions separately into hash tables, and "del" and "nodel" denote whether used VWs are removed from the bags.

terms of collision and error rate trade-off. The overall best method seems to be separate multi-region GMH in combination with deleting used VWs. A curious peculiarity is that for 10,000 VWs the deleting of used VWs seems to have the main impact on the trade-off, while for 1,000 VWs the type of multi-region extension has a greater impact.

Small vocabularies have a higher probability of VWs occurring multiple times. Multi-region extensions only make a difference if the primary VW occurs multiple times. For that reason, multi-region methods without deleting used words do not produce significantly different results than standard GMH for 10,000 VWs. Moreover, intersections between BoVWs are generally larger for small vocabularies since VWs are more likely to appear in an image. The deletion of used VWs has a greater effect if the intersection is already small, so the difference between deletion and no deletion is more pronounced for 10,000 VWs.

Two properties of the results for the vocabulary of size 10,000 are particularly interesting:

- Consider the results for pooled and separate multi-region GMH with deleting used words and sketch size $s = 2$, which is the rightmost point. The point has about 30% of the collision rate of the other methods with their smallest sketch sizes, while introducing virtually no additional error. Compared to the second smallest sketch size of the other methods, we reduce the error rate by approximately 4 percent points, without producing substantially more collisions.

- Pooled and separate multi-region GMH with deleting used VWs and sketch size $s = 2$ are the second rightmost green and turquoise points. They have a lower collision rate and better error rate than the other methods with sketch size $s = 3$.

7.6 Validation Results

In this section, we choose some of the best parameter settings on the tiny set and validate their performance on the ILSVRC2010 validation set.

Since the training set of ILSVRC2010 is a hundred times larger than the training set of the tiny set (cf. Section 5.2.3), we expect increases in collision rates by a factor of 100 as well. We aim for a collision rate of about 1,000 to keep the effort manageable. Thus, we choose parameter settings that optimize the error rate while having a collision rate of roughly 10 on the tiny set.

As a pure standard method without modification, we choose standard GMH. The only setting with a collision rate lower than 10 is sketch size $s = 5$ and 10,000 VWs with about 3 collisions per query. The next higher collision rate of roughly 50 is achieved by $s = 5$ and 1,000 VWs. Since we want to compare to a method without extensions, we refrain from using threshold adaption and choose the vocabulary of size 10,000.

The tf-idf weighted modification of MH with and without permutation grouping yields a collision rate of 5 for 10,000 VWs and sketch size $s = 5$. To move closer to

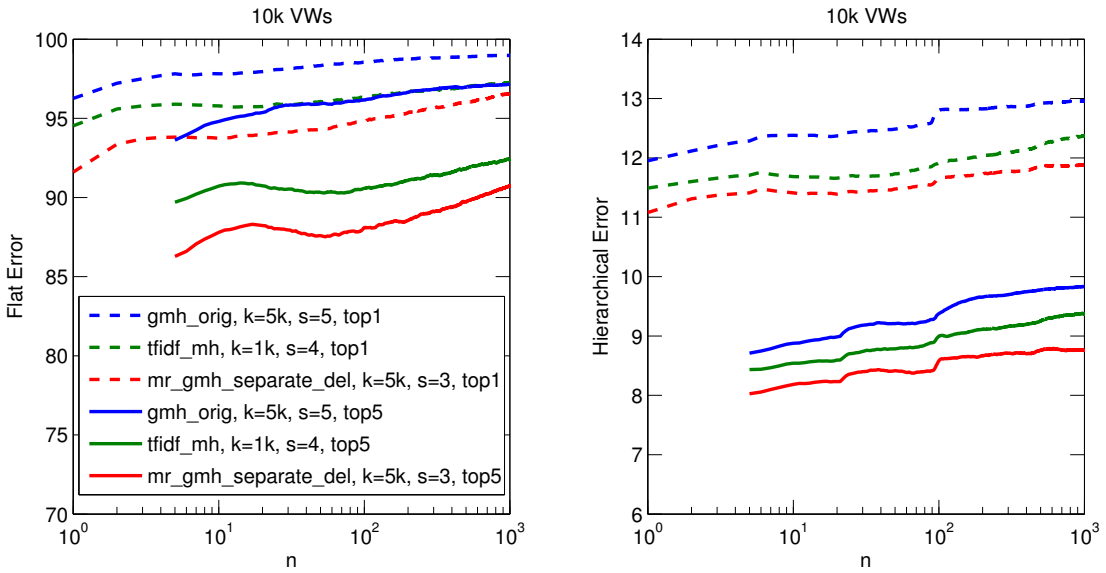


Figure 7.17: Flat and hierarchical error of three methods on the validation set. n denotes the number of neighbors used for categorization.

Table 7.6: Comparison between top 5 flat error rates on the tiny set and on the validation set using 5-NN categorization for different methods. "CR" denotes the collision rate.

Method, parameters	tiny error	tiny CR	val error	val CR
GMH, $k = 5000$, $s = 5$	94.8%	4.0	93.6%	385.6
tf-idf MH, $k = 1000$, $s = 4$	87.0%	61.1	89.7%	3288.0
sep. mr GMH del, $k = 5000$, $s = 3$	81.6%	18.3	86.3%	2156.0

our limit of 10, we decide for tf-idf MH with 10,000 VWs, sketch size $s = 4$ and a threshold adaption to 10,000 VWs to avoid collision rate peaks for queries with many close neighbors.

The GMH modification with the best effort-error rate trade-off is separate multi-region GMH with deletion of used VWs and a vocabulary of size 10,000. Sketch size $s = 3$ yields a collision rate of less than 11.

Figure 7.17 shows the flat and hierarchical error over increasing number of neighbors n used for categorization. All of the methods minimize their flat and hierarchical top 5 error for merely $n = 5$ and their top 1 error for $n = 1$. Figure 7.18 therefore shows the leftmost points of the flat error curves over their collision rates. Table 7.6 shows the corresponding error and collision rates for the same parameters and categorization method on the tiny and the validation set. Note that the tiny set contains only a tenth of the categories and a tenth of training images per category. We still compare results on both sets because we expect a decrease of roughly 18 percent points, which we

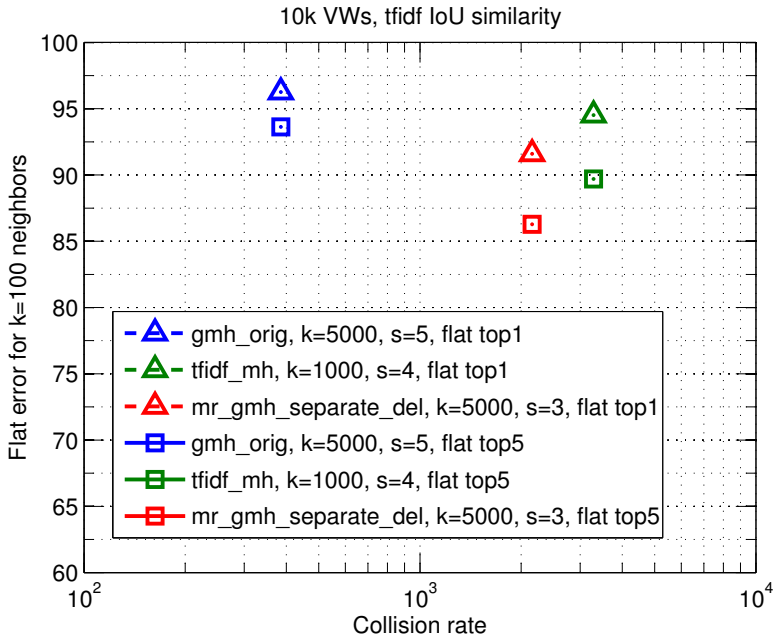


Figure 7.18: Flat error of NN categorization with tf-idf IoU kernel. Top 1 error with 1-NN and top 5 error with 5-NN categorization.

observed for the exact k-NN reference experiments.

- **Standard GMH, $k = 5,000$ sketches of size $s = 5$.**

  The experiment achieves a top 5 error rate of 93.6% with a collision rate of 93.6. Compared to the tiny set, we even see a slight improvement of error rate and an increase by a factor of 100 of the collision rate.

  While we expected such an increase of the collision rate, we rather expected a decrease of error rate, since more categories are likely to make the task more difficult. Instead, the method seems to be able to leverage the additional training images to increase performance.

- **Tf-idf weighted MH, $k = 1,000$ sketches of size $s = 4$.**

  The top 5 error rate of this experiment is 89.7%, which is slightly higher than for the tiny set, while the collision rate increases by a factor of 50. Compared to GMH, the error rate is about 4 percent points lower with ten times the number of collisions.

- **Separate multi-region GMH with deletion of used VWs, $k = 5,000$ sketches of size $s = 3$.**

  This setting accomplishes an error rate of 86.3% with a collision rate of 2156. This is a significant increase of the error rate and again an increase of collision rate by a factor of 100. The results achieve a better error rate with less collisions

than tf-idf MH and an improvement of 7 percent points with an increase of the collision rate by a factor of 6.

The result on the validation set is much closer to the exact k-NN reference than on the tiny set. Exact k-NN is 22 percent points better on the tiny set, which is an improvement of 26%, while the reference is 8 percent points better on the validation set, which is only an improvement of 10%.

The interesting part of the run-time is the query time, because hash table construction is performed offline, whereas ranking and feature extraction are not the at the focus of this work. As the used hash tables require up to 120GB RAM, we ran these experiments on the large machine, which is also used by other people (cf. Section 6.2.1). Thus, timings are not reliable. Measurements range from 6 to 14 seconds per query including IO for loading hashed images and writing collisions.

7.7 Conclusion

After the experiments in this chapter, we are able to provide answers to the questions posed at the beginning of the chapter.

- **How does the vocabulary size affect the performance? Are there preferable vocabulary sizes for categorization?**

  SVMs produce best results for a vocabulary size between 10,000 and 100,000. Categorization with k-NN performs best for 500 and 1,000 VWs.

  For categorization using hash collisions the situation is more complicated. Small vocabularies produce a very high collision rate, so that a useful reduction of collision rate requires large sketches.

- **How do distance measures perform on sparse BoVWs? Is there a best choice? If so, does the preference transfer to the hash collision ranking?**

  The Euclidean distance and IoU similarity produce comparatively weak results. While the Euclidean distance is better for small vocabularies, the IoU similarity improves for larger vocabularies and converges to tf-idf weighted IoU. The Manhattan distance performs consistently better than the IoU similarity.

  The tf-idf weighted IoU similarity outperforms all considered measures. Even if the hashing method, which is used to find neighbor candidates, does not approximate tf-idf IoU, it is better to use this kernel.

- **What is the effect of sketch sizes and the number of sketches? Is there an optimal choice?**

  Larger sketches become more specific and reduce the collision rate exponentially, but increases the quality of collisions. In turn, more sketches increase the collision rate linearly and also improve the collision quality.

There is no optimal choice, but a trade-off between run-time and collision quality. The quality improvement through larger sketch sizes is limited by the absolute number of good collisions, which decreases exponentially. The depletion of good collisions can be compensated with more sketches to some extent, which also improves the collision quality, but increasing the number of sketches also increases the memory usage and query time.

- **Does the quality of collisions correlate with the error rate?**

  While a high collision quality is certainly helpful, it is not high enough to reduce the collision rate without reducing the error rate (with the exception of two variations of multi-region GMH). The main problem is that using the entire training set as neighbor candidates yields a good performance through ranking, even though the quality is low. A smaller candidate set with higher quality usually produces a worse error rate, if it does not retrieve the k nearest neighbors, but offers a better efficiency.

- **To what extent do the extensions improve the trade-off between effort and performance?**

  The tf-idf weighted MH procedure and permutation grouping consistently improved the performance. The tf-idf extension is simpler and achieves greater improvements, while the combination of both methods leads to an additive improvement.

  Threshold adaption does not change the trade-off significantly, but rather provides a method to influence the number of collisions.

  Multi-region GMH improves the trade-off of GMH significantly, especially for a vocabulary of size 10,000.

Chapter 8

Conclusion and Future Work

In this work, we examined the applicability of Min-Hash and Geometric Min-Hash in a categorization context. We evaluated different extensions, including a novel generalization of Min-Hash, called multi-region Geometric Min-Hash. We found that the reduction of nearest neighbor candidates usually also reduces the categorization performance, with the promising exception of two variations of multi-region Geometric Min-Hash, which reduce the number of candidates to about 0.2% while loosing 8 percent points of error rate.

We have evaluated our methods on a publicly available, large-scale database: the ILSVRC2010 dataset, which has a large scale w.r.t. the number of contained images and covered categories.

We hope to offer a starting point for efficient search in large databases for categorization as an alternative to the traditional approach of utilizing more hardware. In contrast to the state of the art, we employ sparse local representations and leverage affine neighborhood information.

Future work should strive towards state-of-the-art results. While this was not our primary goal, there still are a number of directions for further research which were beyond the scope of this diploma thesis.

Using interest region descriptors that incorporate color would be most interesting, as color is an important cue in the ILSVRC2010 dataset. Different classification methods can be applied to collisions to take advantage of dependencies in feature space other than just distances between points. For example, stable classifiers could be employed along the lines of [ZBMM06].

The covered hashing methods are able incorporate corpus-wide word weights. We only experimented with idf weights, but there are more sophisticated possibilities. Roughly 250,000 images in ILSVRC2010 are annotated with bounding boxes. One idea is to weight visual words according to their probability of lying on an object. To be able to use the information present in images without bounding box annotations, it is possible to (a) assume a general prior according to the average throughout all bounding boxes (which turns out to have a Gaussian shape) or (b) obtain an individual estimation for each image by applying an unsupervised objectness measure [ADF10]. It is also

possible to incorporate discriminative weights by calculating the class entropy of each feature. While the objectness probabilities are averaged over the entire database, it is possible to factor in the individual objectness probability of every feature during the computation of the entropy.

Another research direction is a supervised grouping of sketches. For categorization, the main goal is colliding with images of the same class instead of visually similar images. One possibility is a randomized approach that simply minimizes class uncertainty. Another possibility would be dedicating sketches to individual categories or category distinctions.

Certainly, there is room for speed and memory optimizations. For instance, the hash table implementation performs the intersection of collisions of different hash functions within a sketch at query time. This has the advantage that we can modify Min-Hash's sketch size without building new hash tables. Performing the intersection at table construction time obviously decreases the query time, but it can also reduce the memory consumption. The modification reduces the total amount of contents of sketch tables by a factor of $1/s$, while the size of the hash table data structure theoretically increases to N^s, where N is the size of the vocabulary. However, it is possible to employ classic hashing to insert s-tuples of Min-Hash values into a much smaller table. A simple example is $idx = \sum h_i N^i \bmod m$ where m is the size of the smaller hash table and h_i the hash value of the i'th hash value in the sketch.

Glossary

k-NN	k-nearest neighbor.
AMT	Amazon Mechanical Turk.
BoVW	Bag of Visual Words.
CDF	cumulative distribution function.
GMH	Geometric Min-Hash.
ILSVRC2010	ImageNet Large Scale Visual Recognition Challenge 2010.
IoU	Intersection over Union.
LoG	Laplacian-of-Gaussian.
LSH	Locality Sensitive Hashing.
MH	Min-Hash.
SGE	Sun Grid Engine.
SIFT	Scale Invariant Feature Transform.
SVM	Support Vector Machine.
synset	synonym set.
tf-idf	term frequency–inverse document frequency.
VW	Visual Word.

List of Figures

3.1 Decomposition of a second moment matrix 12

4.1 Hash value distribution in a permutation 27
4.2 Effect of image distribution on collision rate 31

5.1 Examples for intra and inter-category variance 45
5.2 Example images from the category “toaster” 46
5.3 Example images from the category “tandem bicycle” and “forklift” . . 47
5.4 Example images from the category “costume” 48
5.5 Hierarchical cost matrix of ILSVRC 50

6.1 Coarse pipeline overview 53
6.2 Feature extraction overview 54
6.3 Hash table construction overview 56
6.4 Hash table query overview 58
6.5 Ranking and categorization overview 58

7.1 Flat error for exact k-NN on validation set 66
7.2 Hierarchical error for exact k-NN on validation set 68
7.3 Flat and hierarchical error for exact k-NN on tiny set 70
7.4 Collision rate of MH on validation set 72
7.5 Collision rate increases linearly with k 72
7.6 Collision rate of MH on tiny set 73
7.7 Collision quality for validation set 74
7.8 Example of collision probability of two parameter settings with same threshold 75
7.9 Flat error of MH over NN parameter 78
7.10 Flat error of MH over collision rate 79
7.11 Flat error of tf-idf MH over collision rate 80
7.12 Histograms of entropy and mutual information of permutations 81
7.13 Flat error of permutation grouping over collision rate 82
7.14 Flat error of threshold adaption over collision rate 83
7.15 Flat error of GMH over collision rate 85
7.16 Flat error of GMH extensions over collision rate 85

7.17 Flat and hierarchical error of GMH, tf-idf MH, and mr GMH over NN parameter on validation set 87
7.18 Flat error of GMH, tf-idf MH, and mr GMH over collision rate on validation set 88

List of Tables

5.1 WordNet statistics 42
5.2 ImageNet statistics 43
5.3 Size of ILSVRC2010 43
5.4 Size of ILSVRC2010 sets 44
5.5 Category detail examples 44
5.6 Tiny set statistics 51

7.1 Error rates for guessing on validation set 64
7.2 SVM error rates on validation set, ImageNet features 65
7.3 SVM error rates on validation set, our features 65
7.4 Runtime for exact k-NN on validation set 69
7.5 Frequency of VWs per image 77
7.6 Flat error and collision rate of GMH, tf-idf MH, and mr GMH on validation set 87

Bibliography

[ADF10] B. Alexe, T. Deselaers, and V. Ferrari. What is an object? In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[BCI08] S. Baluja, M. Covell, and S. Ioffe. Permutation Grouping: Intelligent Hash Function Design for Audio & Image Retrieval. In *IEEE Conference on Acoustics, Speech and Signal Processing*, 2008.

[Bea78] P. Beaudet. Rotationally Invariant Image Operators. In *International Joint Conference on Pattern Recognition*, 1978.

[BHK96] P. N. Belhumeur, J. Hespanha, and D. J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. In *European Conference on Computer Vision*, 1996.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BTVG06] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *European Conference on Computer Vision*, 2006.

[CJ10] O. Chum and J. Large-scale discovery of spatially related images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[CPIZ07] O. Chum, J. Philbin, M. Isard, and A. Zisserman. Scalable Near Identical Image and Shot Detection. In *International Conference on Image and Video Retrieval*, 2007.

[CPM07] O. Chum, M. Perdoch, and J. Matas. Geometric min-Hashing: Finding a (Thick) Needle in a Haystack. In *International Conference on Computer Vision*, 2007.

[CPZ08] O. Chum, J. Philbin, and A. Zisserman. Near Duplicate Image Detection: min-Hash and tf-idf Weighting. In *British Machine Vision Conference*, 2008.

[CVG08] N. Cornelis and L. Van Gool. Fast scale invariant feature detection and matching on programmable graphics hardware. *IEEE Conference on Computer Vision and Pattern Recognition Workshop on Computer Vision on the GPU*, 2008.

[DKN08] T. Deselaers, D. Keysers, and H. Ney. Features for Image Retrieval: An Experimental Comparison. Information Retrieval. In *Vol. 11. Issue 2. Springer*, 2008.

[DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[GHP07] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[Gra07] Kristen Grauman. Pyramid Match Hashing: Sub-Linear Time indexing Over Partial Correspondences. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[Hal04] D. Halperin. Nearest neighbours in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*, chapter 39. CRC Press LLC, 2004.

[KG09] B. Kulis and K. Grauman. Kernelized Locality-Sensitive Hashing For Scalable Image Search. In *International Conference on Computer Vision*, 2009.

[LLS08] B. Leibe, A. Leonardis, and B. Schiele. Robust Object Detection with Interleaved Categorization and Segmentation. *International Journal of Computer Vision*, 2008.

[LLZ+11] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang. Large-Scale Image Classification: Fast Feature Extraction and SVM Training. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[Low04] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 2004.

[LSP06] S. Lazebnik, C. Schmid, and J. Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[LWZ+08] X. Li, C. Wu, C. Zach, S. Lazebnik, and J-M. Frahm. Modeling and Recognition of Landmark Image Collections Using Iconic Scene Graphs. In *European Conference on Computer Vision*, 2008.

[MS04] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, 2004.

[MTJ07] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Neural Information Processing Systems*, 2007.

[NS06] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[OPM02] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[OT01] A. Olivia and A. Torralba. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *International Journal of Computer Vision*, 2001.

[PCI+07] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[PCM09] M. Perd'och, O. Chum, and J. Matas. Efficient Representation of Local Geometry for Large Scale Object Retrieval. *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[RU11] A. Rajaraman and Jeffrey D. Ullman. Mining of Massive Datasets, 2011. `http://infolab.stanford.edu/~ullman/mmds.html`.

[SB91] M. J. Swain and D. H. Ballard. Color Indexing. *International Journal of Computer Vision*, 1991.

[SC00] B. Schiele and J. L. Crowley. Recognition without Correspondence using Multidimensional Receptive Field Histograms. *International Journal of Computer Vision*, 2000.

[SZ03] J. Sivic and A. Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *International Conference on Computer Vision*, 2003.

[TP91] M. A. Turk and A. P. Pentland. Face Recognition Using Eigenfaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1991.

[vdSGS08] K. E. A. van de Sande, T. Gevers, and C. G. M. Snoek. Evaluation of color descriptors for object and scene recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[VSWB06] J. Vogel, A. Schwaninger, C. Wallraven, and H. H. Bülthoff. Categorization of natural scenes: Local vs. global information. In *Proceedings of the 3rd Symposium on Applied Perception in Graphics and Visualization*, 2006.

[VZ05] Manik Varma and Andrew Zisserman. A Statistical Approach to Texture Classification from Single Images. *International Journal of Computer Vision*, 2005.

[WHL10] T. Weyand, J. Hosang, and B. Leibe. An Evaluation of Two Automatic Landmark Building Discovery Algorithms for City Reconstruction. In *European Conference on Computer Vision Workshop on Reconstruction and Modeling of Large-Scale 3D Virtual Environments*, 2010.

[YZG09] K. Yu, T. Zhang, and Y. Gong. Nonlinear Learning using Local Coordinate Coding. In *Neural Information Processing Systems*. 2009.

[ZBMM06] H. Zhang, A. C. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[ZYZH10] X. Zhou, K. Yu, T. Zhang, and T. S. Huang. Image Classification using Super-Vector Coding of Local Image Descriptors. *European Conference on Computer Vision*, 2010.