Technische Universität Berlin

# Object Classification using Local Image Features


| | |
|---|---|
| Author | Sebastian Nowozin |
| Supervisor | Marc Jäger |
| Research direction | Computer Vision and Remote Sensing |
| Document date | May 8, 2006 |

Die selbständige und eigenhändige Anfertigung versichere ich an Eides statt.

Berlin,

_____

*to Juan,*

*and to my parents, Annette and Reinhard*

## Zusammenfassung

Die Zuordnung von auf digitalen Bildern dargestellten Objekten in abstrakte Objektklassen ist eine der zur Zeit schwierigsten Aufgaben im Bereich der Computer Vision. In den letzten zehn Jahren wurden leistungsfähige Methoden entwickelt, die es erlauben markante Merkmale in Bildern zu finden und zu beschreiben. Um jedoch Verfahren des maschinellen Lernens auf diese Merkmale anwenden zu können ist eine darauf aufbauende Repräsentation erforderlich.

*Merkmalsmengen* werden hierfür erfolgreich eingesetzt. Als maschinelle Lernmethode werden oft Support Vektor Maschinen zur Klassifizierung eingesetzt. In dieser Arbeit werden die wichtigsten Ansätze in diesem Kontext untersucht und ihre Defizite erörtert.

Um diese Defizite zu überwinden schlagen wir vor, *Merkmalsmengen* als einen Spezialfall von *Merkmalsgraphen* zu betrachten. Diese Betrachtungsweise erlaubt es uns zusätzliche relevante Informationen *zwischen* einzelnen Merkmalen in die Repräsentation zu integrieren. Dazu bedienen wir uns der Metainformationen der Merkmale, wie zum Beispiel der Position, Skala, Orientierung und Form der Merkmalsregion. Sorgfältiges Vorgehen erlaubt es uns dabei die Invarianzen der Merkmalsextraktion bei zu behalten.

Um diesen neuen Ansatz zu validieren benutzen wir eine standardisierte Untermenge des ETH-80 Referenzdatensatzes zur Evaluation von Objektklassifizierungssystemen.

## Abstract

Object classification in digital images remains one of the most challenging tasks in computer vision. Advances in the last decade have produced methods to repeatably extract and describe characteristic local features in natural images. In order to apply machine learning techniques in computer vision systems, a representation based on these features is needed.

A *set of local features* is the most popular representation and often used in conjunction with Support Vector Machines for classification problems. In this work, we examine current approaches based on set representations and identify their shortcomings.

To overcome these shortcomings, we argue for extending the set representation into a *graph representation*, encoding more relevant information. Attributes associated with the edges of the graph encode the geometric relationships *between* individual features by making use of the meta data of each feature, such as the position, scale, orientation and shape of the feature region. At the same time all invariances provided by the original feature extraction method are retained.

To validate the novel approach, we use a standard subset of the ETH-80 classification benchmark.

# Contents

# List of Figures

# Chapter 1

# Introduction

In this chapter we determine the context of this work and motivate the goal we want to achieve. The goal is precisely described and an outline prepares the reader for the following chapters.

## 1.1 Motivation

Ever since their invention, computer systems have become more powerful, more intelligent and more prevalent in our everyday life; so much that in the long term we can not imagine a life without the technology powered by computers. Yet, the usefulness of current and upcoming technology will be determined not so much by the further improvement of the computational capability or physical characteristics, but by the ability of systems to interact with and work for humans; simply said, in order to improve, systems will have to act intelligently.

Probably the most important requirement to act intelligently is the ability to process and reduce a large amount of information quickly to the few relevant details. The *human vision* system is nothing short but impressive in this regard. Eyesight is the most important human sensory input because it is so helpful to make sense from the real world. Hence, to successfully model the human vision capability in computer systems promises tremendous rewards for practical applications.

*Computer vision* is the research field trying to harvest this reward. It has its origins in the 1970'ies and continues to grow as research field due to the large scale availability of digital cameras since the 1990'ies. While some tasks have been addressed successfully – such as optical character recognition (OCR), face detection, vehicle and object tracking – some tasks simple for every human still puzzle even the best computer systems. One such task is *object classification*, the central problem of this thesis' work. In a simple object classification task, a single object is shown on an image. The task is to reduce the information in this image to a single number, determining the object's class. Here class means an abstract class like "the class of all bananas". Hence, if an image shows a banana, the object classification system shall assign the banana class value to it. While there remains a large ambiguity what constitues a valid class and where membership starts and stops or whether membership is a fuzzy concept, a more interesting question is why all humans can handle in-class variations in the same, repeatable and confident fashion.

The most recent approaches to object classification first transform the image into another representation, reducing the information such that the remaining information consists of

*features*, each of which is invariant to some important variances of visual perception such as brightness, contrast and position within the image, abstracting away some aspects not related to the object. However, the variance in appearance of objects within the same class remains. In this diploma thesis we will describe the most recent approaches to object classification based on such features in detail and will try to identify their common shortcomings.

We motivate this thesis by the long term requirement of increasing the capability of computer systems to deal with the real world: on a coarse level, computers are more useful if they can make sense of the real world; on a smaller level, improving the current solutions to the object classification problem is a good test for that higher level goal.

## 1.2 Problem definition and goals

The *problem* this thesis is concerned with is object classification based on modern local image features and kernel methods. More precisely, the problem is to build a system, that can be decomposed into the following parts.

- *Object classification.* The system shall be able to *learn* abstract object classes from labelled training images shown to the system.
- *Modern local image features.* The information from the image shall be processed using vectorial, local image features, not by directly accessing the images on a per-pixel basis.
- *Kernel methods.* The learning part of the system shall only access the training and testing samples through the use of a positive-definite kernel function.

For this thesis' work, the goals to be achieved are the following.

- To give an overview of current kernel based approaches to object classification from images.
- To identify common shortcomings of current set of keypoints based approaches.
- To propose a model – *graphs* of local image features – that could overcome these shortcomings by incorporating more relevant information into the representation.
- To evaluate and analyze the proposed model.
- To provide conclusions from the results and give directions where further research is necessary.

## 1.3 Overview of chapters

In **chapter 2** the theoretical background of this thesis's work is explained in four sections. First, in section 2.1 the problem of object classification is introduced and the background on digital image processing and computer vision is given. Second, in section 2.2 an overview of modern local image features is provided. The Gaussian scale-space is introduced as basic tool to represent features occurring at multiple scales. In the following subsections the interest point detectors and descriptors we use are introduced with a focus on the Scale-Invariant

Feature Transform (SIFT). In the third part, section 2.3, Support Vector Machines are introduced through statistical learning theory. All important machine learning algorithms used in the overall object classification system are described in detail. Finally, in the fourth section 2.4, an important kernel with respect to our system – the Marginalized Graph Kernel – is described in detail.

In **chapter 3** we discuss one common paradigm of object classification using local features, the *set of keypoints* model. We analyze around a dozen recent publications on the topic and conclude with a summary of the advantages and disadvantages of the individual approaches.

In **chapter 4** we introduce our novel approach by picking up the conclusions of the previous discussion. We introduce some novel ideas how to improve over set based classification approaches. Motivated by the availability of meta information for the image features – such as position, size and orientation within the image – we introduce *edge attributes* and an *edge kernel*, which capture and compare this information between feature pairs.

To validate and evaluate our approach, we propose and carry out a number of experiments in **chapter 5**. To examine the different effects of the components in the proposed system, we first perform a parameter selection and evaluation on only the edge kernel and different feature types. The overall system is then evaluated, followed by some baseline experiments.

Concluding this work, in **chapter 6** a summary of the proposed system is given, together with identified strengths and weaknesses. Possible directions of further work are presented.

## 1.4 Acknowledgments

I would like to address my special thanks to Marc Jäger for the enthusiastic supervision and insightful discussions; even while having to finish four conference papers he always had time to take a look at my problems. Further I would like to thank Gökhan Bakır for giving just the right references, hints and advice at the right time. Finally, I thank Professor Olaf Hellwich for accepting the proposal for this thesis and providing the right environment, as well as letting me burn thousands of experimental CPU hours on spirit.

# Chapter 2

# Theoretical Background

This chapter provides all the necessary background to this thesis' work. The main cornerstones are modern local image features and kernel methods. For the former, we will examine the popular Scale-Invariant Feature Transform (SIFT) method. The latter topic will be addressed in detail for our use of Support Vector Machines (SVM) for classification and to detail a popular kernel function defined on graphs, the Marginalized Graph Kernel (MGK).

## 2.1 Object classification

In this section we define our problem within the computer vision field and give the necessary definitions for the following in-depth literature discussion.

### 2.1.1 Computer Vision

In the most general context, computer vision systems create a model of the world from digital images [23]. The scope of the model depends on the task the computer vision system aims to solve and can be predefined by the designer of the system or partially be learned from the available data. Most often computer vision systems can be decomposed into the following three parts.

1. A *data aquisition part*, which uses sensors or external interfaces to aquire one or more digital images. The data aquisition can also include Digital Image Processing techniques to preprocess the data, such as removing irrelevant noise.

2. The *representation of data*, arguably the most important part of the system. The aquired data is transformed into an alternative representation more suitable to the problem: information irrelevant to the problem is supressed or discarded, while information likely to be useful is kept or amplified. Methods extracting useful pieces of information, so called *feature extractors* are commonly used. If the representation is good, the remaining part of the system is trivial. Hence, the largest part of the computer vision literature deals with suitable representations and how to extract them from a digital image.

3. A *decision part*, achieving the system goal from the represented data. For complex tasks such as object recognition or face detection these often include state of the art techniques

from the field of Machine Learning. Often probable hypotheses are formulated and verified.

Although modern computer systems have a vast amount of computing power and this capacity continues to increase exponentially, most tasks that are natural and easy to every human, such as recognizing or classifying objects shown on pictures turn out to be extraordinary difficult to computers. Conversely, this highlights the astonishing ability the human vision system is capable of: we are able to reason about a wide variety of highly abstract concepts and patterns in the scenes we perceive; with no apparent difficulty we are able to connect our vast knowledge instantly with what we perceive, generalizing where possible, specializing where necessary; constantly learning and consistently integrating the new information from our environment with our model of the world. Although the field of computer vision has in many parts matured over the last 20 years, it is still not possible to say whether we will ever be able to build systems that will match or exceed the general capability of the human vision system.

### 2.1.2 Object classification

We now give a definition of "object classification" as a basic problem in computer vision. The related "object recognition" is defined as well, in order to avoid confusion when discussing the literature.

**Object recognition** is the process of detection and classification of an object *previously seen* or learned by a system. This includes the cases where an object has previously only been seen from a different perspective.

**Object classification** is the detection and classification of any object, which is a member of a given set of abstract classes into one such abstract class. The important difference here is two fold; first, any object, even previously unseen objects shall be classified correctly, and second, the *object class* is an abstract class such as "car", "fruit", etc. In general, object classification is a more difficult task than object recognition.

Both problems are similar because they have to relate previous knowledge about objects or object classes to new data. Trivially, object recognition could be seen as an object classification task where each object constitues its own class. But were object recognition systems only have to be invariant against changes in pose, clutter and illumination, object classification systems further need to address inter-class variances of the objects characteristics, such as shape, color and size.

## 2.2 Local image features

In this section we examine modern local image features for natural images. For our approach to object classification local image features are the only input to the classification system. We first define the role of a feature and divide features for computer vision systems into global and local features. The importance of scale is highlighted and the scale-space is introduced. To illustrate the abstract discussion, we then discuss in detail the recently most popular local image feature, the Scale-Invariant Feature Transform (SIFT).

### 2.2.1 Introduction

For any computer vision system the notion of a "feature" is important. Computer vision systems extract features from their input data and produce useful information from those features. Castleman [7] defines a *feature* as follows.

> "A *feature* is a function of one or more measurements, computed so that it quantifies some significant characteristic of the object."

This very general definition does not specify how exactly a feature is computed nor if it describes the whole object to be measured or a part of it. This distinction is made in a further separation into *global features* and *local features*. Global features quantify characteristics of the whole image to be measured. An example would be a color histogram of an image which gives important information about the whole image.[1] Local features quantify characteristics of a particular region of the object to be measured. For the case of images, a local feature may be limited to a small spatial area.

Why would one prefer one over the other? The answer depends very much on the problem to be solved. For the problem we are concerned with in this thesis, object classification, it is advantageous to chose local image features for the following reasons.

- Object to feature relationship.

  A global feature is a function of the entire image. As we are concerned with one object within the image at a time, it would be optimal to ignore any information not related to the object of our concern, such as the image background or other objects. In general this is not possible, as the problem of foreground-background segmentation is non-trivial. However, by using local image features we can design image features in such a way that a single feature is unlikely to cover an area of the image that belongs to both the foreground object we are interested in *and* the background we want to ignore. Given a set of such extracted features, we do not know which features belong to the object and which do not, but we can be quite certain that each feature is either a feature from the foreground object or the background. This partitioning of primitive features can be exploited to our advantage in our problem setup.

- Invariance against spatial transformations.

  The object classification task we are trying to solve should naturally be solved in a way such that moving the object within the image or rotating it should not affect the classification result. In general, we want our system to be *invariant* against spatial transformations. Using some global features this can be achieved rather easily for some variances, such as translation and rotation in the view plane. But for other variances, such as enlargement of the object, it can be difficult to design a global feature to produce invariant values. For local image features techniques have been developed to extract features invariant of translation, rotation and changes in scale, while also being robust to general affine transformations. We will discuss this in detail in section 2.2.4.

- Relationship between features.

  While a global feature condenses characteristics of an object to be measured into one piece of information, for local features not only the single feature itself is important but

[1] For example, Chapelle et al. [9] use such global feature for image classification.

it may be the case that the *relationship between features* gives more useful information about the object than any single feature alone.

For local image features, it might be the relative orientation of some features that is important for the system. For example, estimating the direction a person is looking at in an image could be solved using the relationship of the person's facial features, such as eyes, nose and mouth. It would be difficult to estimate the direction from any single feature alone.

We will use this idea in chapter 4 to derive an approach for the problem of object classification.

If local features or any local structures are analyzed within digital images, the notion of *scale* is crucial. It is the relative scale of a structure present in the image that separates it from being a detail or a fundamental structure in the image. Additionally, the scale can imply a hierarchy of the features, that is, it is structuring the primitive features. How this hierarchy relates to the computer vision problem at hand depends very much on the problem itself, but such a natural hierarchy almost always contains helpful information.

Because of its importance, scale has been modeled even in very early computer vision algorithms. Lindeberg [28] cites quadtree's in the early 1970'ies and image pyramids in the early 80'ies as examples, which are now covered in any introductory book on digital image processing [49, 18, 7]. But even among recent methods there is a wealth of methods operating at multiple scales, such as wavelet decompositions and multi-grid methods.

A very general and powerful approach to model scale in a digital image has been proposed by Witkin [57]. We will discuss his model, the so called *scale-space* in detail in the next section, as most recent local image feature extraction methods are based on it.

### 2.2.2 Gaussian scale-space

Following Lindeberg [28] a *scale-space* representation embeds a signal $f : \mathbb{R}^D \to \mathbb{R}$ into a space $L : \mathbb{R}^D \times \mathbb{R}_+ \to \mathbb{R}$, parametrizing $f(\cdot)$ with a *scale parameter* $t$ such that $L(\cdot; 0) := f(\cdot)$ and $L$ is the solution of the equation

$$\partial_t L = \frac{1}{2}\nabla^2 L = \frac{1}{2}\sum_{i=1}^{D} \partial_{x_i x_i} L. \tag{1}$$

Subject to some reasonable constraints, namely

- *causality*, also called *non-creation property*, meaning no new zero level surface must be created as the scale parameter $t$ increases, such that no new local extrema can be created,
- *isotropy* and *homogeneity*, unbiased processing across all directions and scale levels,

it can be shown that for 1D signals the Gaussian kernel $g : \mathbb{R}^D \times \mathbb{R}_+ \setminus \{0\} \to \mathbb{R}$,

$$g(\cdot; t) = \frac{1}{(2\pi t)^{\frac{N}{2}}} e^{-\sum_{i=1}^{N} \frac{x_i^2}{2t}} \tag{2}$$

with $\sigma = \sqrt{t}$ is unique in defining a scale-space $L(\cdot; t) = g(\cdot; t) * f(\cdot)$ fulfilling all the constraints and solving equation (1).

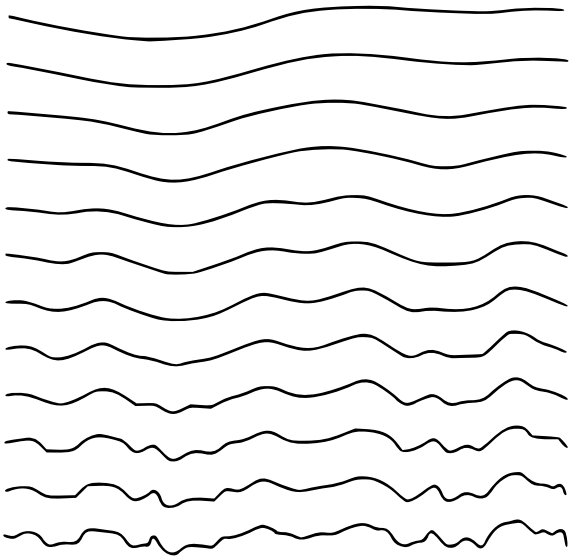

Figure 2.1: Example of a 1D Gaussian scale-space $L$: the original signal (bottom) is incrementally smoothed to produce more and more coarse versions (top). The diagram is from [57, 28]

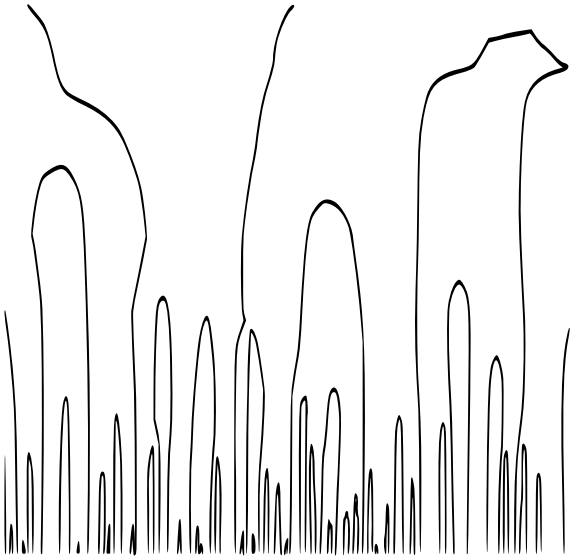

Figure 2.2: The zero-crossings of the second spatial derivative of the signal form paths; their number is reduced as the scale parameter grows, without new paths being introduced. This holds for any order of spatial derivatives.

This does not generalize to higher dimensions, namely the *causality* constraint breaks down and new local extrema could be created as $t$ increases. Still, under considerations from mathematical physics it can be shown the Gaussian kernel is unique in defining a reasonable scale-space. An in depth discussion with proofs is Lindeberg [28], but for practical use just assume the non-creation property also holds for higher dimensional signals.

An important result from the non-creation property is concerned with the zero-crossings in $L_x$, the spatial derivatives of $L$: the zero-crossings in $L_x$ form closed curves (for 1D signals) or closed surfaces (for higher dimensions) in the scale-space.

This is illustrated in figure 2.1 and 2.2 adapted from [57]. The detection of extrema points in the spatial derivatives of the scale-space $L$ will be the basis for local image feature extraction algorithms. The normalization necessary to detect extrema points over scale will be discussed in the next section.

Another concern is the discretization of the scale-space, which is problematic as some concepts of differential geometry, such as level curves, have no natural counterpart in the discretized space. This is discussed in [28] and will not be repeated here as it does not add much to the understanding of the concepts involved.

#### Scale-normalization

In section 2.2.2 we have seen that the signal magnitude at each point in scale-space decreases as the scale parameter $t$ increases. Lindeberg [29] proposes a *normalized derivative operator* as

$$\partial_{\gamma-\text{norm}} := t^{\frac{\gamma}{2}} \partial_x,$$

which depends on a normalization parameter $\gamma$. Here, we will only consider $\gamma = 1$, which results in *perfect scale invariance*, but in [29] other values of $\gamma$ are analyzed. For $y = 1$ the derivative operator becomes

$$\partial_{\gamma-\text{norm}} := t^{\frac{1}{2}}\partial_x = \sqrt{t}\partial_x = \sigma\partial_x.$$

Normalizing the differential operators allows one to search for extrema points of differential expressions over scale. Such extrema points express the existence, location and approximate size of characteristic features of the signal. Depending on the kind of differential expression used, we can examine the scale-space for different features. For 2D images an interesting feature is a "blob", a region that contrasts in brightness with its surrounding region, for example a filled white circle on a dark background. Lindeberg [29] proposes to examine expressions on the normalized Hessian matrix $\mathcal{H}$ as

$$\begin{aligned} \text{trace}\, \mathcal{H}_{\gamma-\text{norm}} L &= t^{\gamma}\nabla^2 L = t^{\gamma}(L_{xx} + L_{yy}), \\ \det \mathcal{H}_{\gamma-\text{norm}} L &= t^{2\gamma}(L_{xx}L_{yy} - L_{xy}^2). \end{aligned} \tag{3}$$

In the next section, we will see how equation (3) is used in the SIFT feature extraction algorithm.

### 2.2.3 Affine co-variant features

The scale-normalized Laplacian-of-Gaussian filter response (3) is isotropic and responds to circular blobs. This works well to detect the same feature points under translation, rotation and uniform scale changes of an image. In [37], Mikolajczyk and Schmid analyzed the general case for *affine* transformations. For isotropic filters, two problems occur:

1. Unstable localization property.

   Even under general affine transformations that involve a large change of viewpoint relative to the interest point's surrounding surface the Laplacian-of-Gaussian filter response has extrema at the points position. However, the position of these extrema points shift with affine transformations, making the position unstable.

2. Descriptor window.

   The neighborhood of an interest point is normally used to compute a fixed size interest point descriptor. For affine transformations the circular spatial neighborhood of a point changes. In case a circular region is used, two identical interest points in two images can produce two different descriptors.

To counter these shortcomings of the isotropic LoG filter, Mikolajczyk [37] introduces two methods, i) a multi-scale Harris-Laplace interest point detector more robust than the normal LoG detector, and ii) an estimation of the affine shape of a local structure based on the second moment matrix. Although there have been quite similar approaches been used as early as 1994, compare Lindeberg and Gårding [30, 31], Mikolajczyk summarizes these results into a practical algorithm for general interest point detection.

### Multi-scale Harris-Laplace detector

The Harris corner detector [22] is the most famous corner detector. It uses a *cornerness* measure on the *second moment matrix* to judge whether a point in the image is a corner or not.

The second moment matrix describes the local image structure around a point $\mathbf{x}$. Mikolajczyk [37] extend the definition to obtain the scale-adapted second moment matrix

$$\mu(\mathbf{x}, \sigma_I, \sigma_D) = \begin{bmatrix} \mu_{11} & \mu_{12} \\ \mu_{21} & \mu_{22} \end{bmatrix} = \sigma_D^2 g(\cdot; \sigma_I) * \begin{bmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x L_x(\mathbf{x}, \sigma_D) \\ L_x L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}. \tag{4}$$

Here, $\sigma_I$ is the *integration scale* and $\sigma_D$ is the differentiation scale, thus $\mu(\mathbf{x}, \sigma_I, \sigma_D)$ describes the distribution of gradients around the point $\mathbf{x}$. The cornerness measure $c(\mathbf{x}, \sigma_I, \sigma_D)$ of a points $\mathbf{x}$ is then defined as

$$c(\mathbf{x}, \sigma_I, \sigma_D) = \det\left(\mu(\mathbf{x}, \sigma_I, \sigma_D)\right) - \alpha \operatorname{trace}^2(\mu(\mathbf{x}, \sigma_I, \sigma_D)).$$

Maxima in $c$ describe corner points in the image.

In practice the Harris-Laplace detector then works as follows. First, interest points are collected by detecting extrema in the 8-neighborhood of the scale-normalized LoG filter response planes. Let $\sigma_I$ be the scale the extrema at $\mathbf{x}$ was detected. Second, the local LoG extrema over scale for each point $\mathbf{x}$ is determined by searching the adjacent LoG planes at the same spatial location. In case no maximum is found, the point is discarded. Third, the nearest spatial location in the neighborhood of $\mathbf{x}$ that maximizes the Harris cornerness measure is located. Fourth, if any change in location or scale has occurred in step two or three, repeat from the second step.

### Affine shape estimation

To estimate the affine shape surrounding an interest point, Mikolajczyk [37] uses an extension of the second moment matrix (4) which uses anisotropic Gaussian kernels. The *affine second moment matrix* is defined as

$$\mu(\mathbf{x}, \Sigma_I, \Sigma_D) = \det\left(\Sigma_D\right) g(\cdot; \Sigma_I) * \left((\nabla L)(\mathbf{x}, \Sigma_D)(\nabla L)(\mathbf{x}, \Sigma_D)^\top\right). \tag{5}$$

$\Sigma_I$ and $\Sigma_D$ are the covariance matrices for the Gaussian integration and differentiation kernels, respectively. To simplify the computation we set $\Sigma_I = s\Sigma_D$, where $s \in \mathbb{R}_+$. Upon this affine second moment matrix $\mu$, an *isotropy measure* can be defined as the ratio $\mathcal{Q}$ of the smallest eigenvalue to the larger eigenvalue as

$$\mathcal{Q}(\mu) = \frac{\lambda_{\min}(\mu)}{\lambda_{\max}(\mu)}.$$

$0 < \mathcal{Q}(\mu) \leq 1$ measures the isotropy of $\mu$, where $\mathcal{Q}(\mu) = 1$ corresponds to a perfectly isotropic shape around the interest point. While the exact algorithm for establishing the affine transformation is rather technical, the fundamental idea is to proceed as follows. First, the Harris-Laplace detector is used to extract isotropic interest points. Second, for each point, $\Sigma_D$ and $\Sigma_I$ is chosen as to maximize $\mathcal{Q}(\mu)$. Third, the point is possibly shifted if a larger Harris measure is found in the neighborhood. Fourth, the process iterates from the second step, if the point position and shape has not converged or diverged yet. Divergence can

happen due to noise or elongated structures; convergence conditions have been examined by Lindeberg and Gårding [31].

After convergence the affine shape around the point has been estimated and the level sets of the anisotropic Gaussian implicitly define elliptical closed curves around the point. One fixed level is selected and the ellipse defines the region the descriptor is created from. For practical reasons, the image content in the ellipse is mapped to a circular region and the orientation of this region is normalized by gradient orientation histograms. The descriptor creation can be carried out using any region descriptor, such as SIFT, PCA-SIFT, Jets, etc.

Summarizing, the approach reduces the discriminating power of the resulting descriptors for the benefit of relative invariance against affine transformations. If the problem to be solved involves recognition using large affine transformations, the approach leads to substantial improvements. We will use the binaries provided by Mikolajczyk[2] to extract the features for our object classification system.

In the next section we examine SIFT, the most popular region descriptor and its original scale-space approximation.

### 2.2.4 Scale-Invariant Feature Transform (SIFT)

One of the most popular local image feature for general natural images is SIFT – the Scale-Invariant Feature Transform – which was developed in 1999 by David Lowe [32], and later refined and extensively described in [33]. In this section we examine SIFT in detail as it provides the features we will use to learn object categories from.

The SIFT algorithm is based directly on the scale-space framework given by Lindeberg [28], but extends the idea of locating interest points in scale-space to also describe their characteristics in a way so the resulting descriptor is invariant or robust against changes in scale, rotation and affine transformations. SIFT consists of three parts.

1. Efficient discretization of the Gaussian scale-space.
2. Feature localization.
3. Construction of an invariant feature descriptor.

We examine each part in detail as they are separable.[3]

#### Efficient discretization of the Gaussian scale-space

The Gaussian scale-space discussed in section 2.2.2 is based on a continuous mathematical description. To implement scale-space algorithms on computer hardware this continuous space needs to be discretized. Using a fine discretization requires more computational and storage resources but approximates the ideal mathematical scale-space better than a coarse discretization. In SIFT, a hierarchical way of approximating the Gaussian scale space is chosen based on the following two observations.

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

[3] In fact, in later works [39, 38] the idea of an integrated feature extraction method is refined by considering the localization of an interesting region in the image and the concise description of such regions as two separable steps.

1. Incremental convolution.

   Convolving an image incrementally by a Gaussian filter $g(\cdot;\sigma)$ with small $\sigma$ is computationally less expensive than a single convolution with a large value $\sigma$.

   Convolution of a 2D image $I(x;y)$ with a Gaussian $g(\cdot;\sigma)$ is usually implemented separably as two 1D Gaussian convolution passes. Each 1D convolution is carried out on a fixed sized *support* of a discretized Gaussian *kernel*. Usually, for a given $\sigma$, an odd integer support larger than $3\sigma$ is chosen. For example, given $\sigma = 2$, the support size would be seven or more. The computational demand scales linear with the size of the support, hence linear with $\sigma$.

   For two Gaussian filters $g(\cdot;\sigma_1)$, $g(\cdot;\sigma_2)$ the following relation holds:

   $$g(\cdot;\sigma_1) * g(\cdot;\sigma_2) = g\left(\cdot;\sqrt{\sigma_1^2+\sigma_2^2}\right) \tag{6}$$

   Recursively, for a given large $\sigma$, computational effort can be saved by repeated convolution with a small $\sigma$ value, obtaining the same result.

2. Coarsening of discretization.

   Given an image $I_1$ known to have been convolved using a Gaussian $g(\cdot;\sigma)$, such that $I_1 = g(\cdot;\sigma) * I$, then if the image $I_1$ is convolved to obtain $I_2$ such that $I_2 = g(\cdot;2\sigma) * I$ holds, then the image $I_2$ can be sampled at half the resolution in each dimension without loss of information.

   Given only $I_1$ and $\sigma$, how does one chose $\sigma'$ in

   $$I_2 = g(\cdot;\sigma') * I_1,$$

   such that $I_2 = g(\cdot;2\sigma) * I$ holds?

   From equation (6) we immediately obtain $\sigma' = \sqrt{3}\sigma$, as $\sqrt{\sigma'^2+\sigma^2} = \sqrt{3\sigma^2+\sigma^2} = 2\sigma$.

To obtain a discretized scale space, we incrementally convolve an input image with Gaussian filters. The scale discretization is not done in equidistant steps like the spatial dimensions, but discretized so that each discretization step is separated from the next by a constant factor. The reason is, that a constant factor will provide the necessary normalization with $\sigma^2$ when we approximate the Laplacian-of-Gaussian filter $\nabla^2 g$ in a later step. But it should not concern us now.

Assume the input image $I$ is known to have been convolved with a given $\sigma$. The filters are chosen such that after a fixed number of convolutions we obtain an image convolved with $2\sigma$. All images from $\sigma$ to $2\sigma$ constitute an *octave*.

The last image of the octave - $I$ convolved with $2\sigma$ - is downscaled to one fourth its size by halving each dimension. The downsized version is the input to the next octave. This incremental process is shown schematically for one octave in figure 2.3.

Each octave goes from $\sigma$ to $2\sigma$ and holds a fixed number of $s$ planes. While the construction has $I_n = g(\cdot;k^n) * I$, where $k$ is the constant factor, only the image $I_0$ and $\sigma$ is the input to the octave, so we do not explicitly know $I$. Let $k = 2^{\frac{1}{s}}$ be the constant factor, then obviously $k^s = 2$.

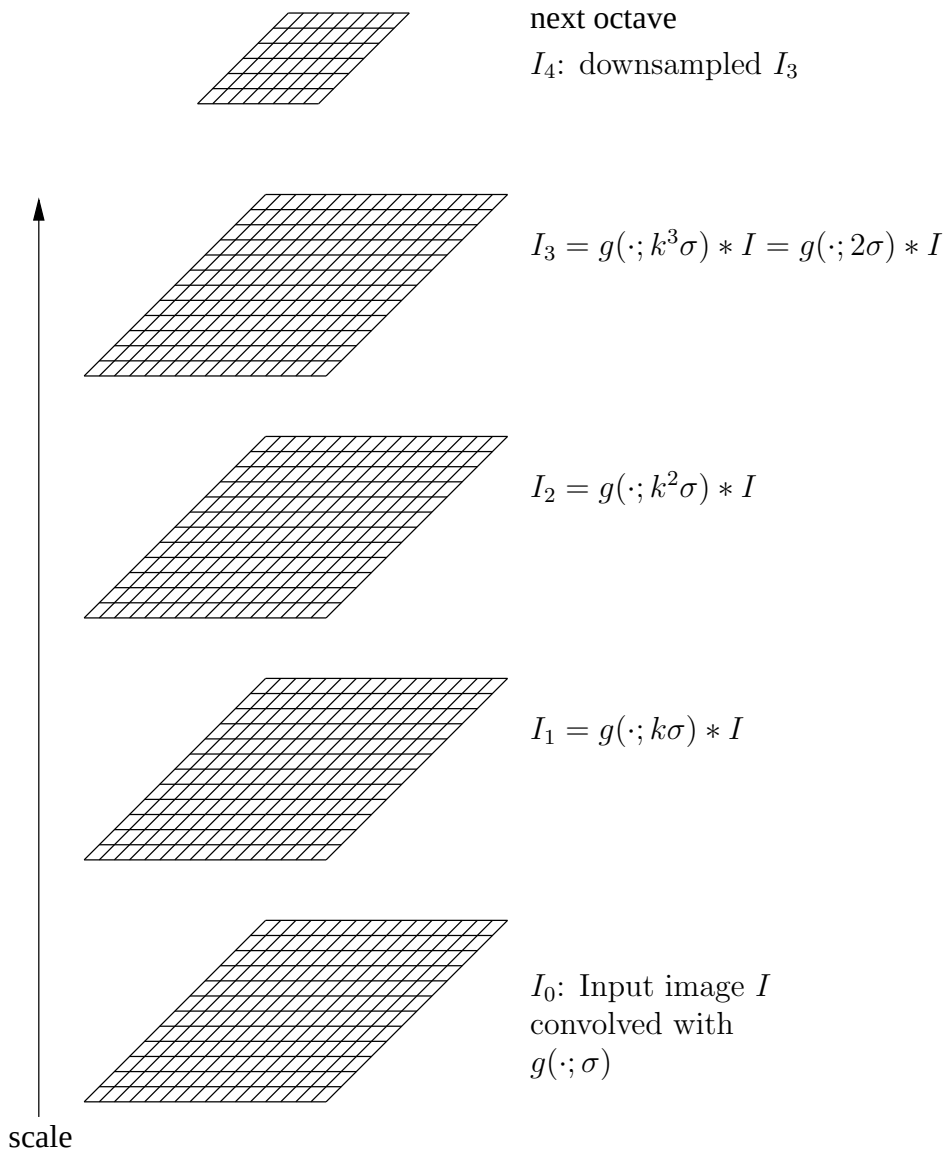


Figure 2.3: Efficient Gaussian scale-space discretization by octaves. Shown is one octave, with an input image $I_0 = g(\cdot; \sigma) * I$ as starting plane of the octave. The image is incrementally convolved to obtain $I_1$, $I_2$ and $I_3$, so that each image has $I_p = g(\cdot; k^p\sigma) * I$. Here $k = 2^{\frac{1}{s}}$ with $s = 3$ the number of new planes in the octave (the number of total planes minus one), so $k^s = 2$.

To chose $\sigma_p$ in

$$g(\cdot; k^{p+1}\sigma) = g(\cdot; \sigma_p) * g(\cdot; k^p\sigma),$$

Table 2.1: An example incremental convolution in a scale-space octave: an original image $I$ is known to have been convolved with $\sigma = 1.5$. How to incrementally convolve it such that, i) each scale's filter sigma is separated by a constant factor relative to $I$ and ii) after $s = 3$ convolutions we obtain an image $I_3$ convolved with $2\sigma = 3.0$.

| $p$ | Input image | equals $I$ convolved with | $\sigma_p$ |
|---|---|---|---|
| 0 | $I_0$ | $g(\cdot; 1.5)$ | $\sigma_0 = k^0\sqrt{k^2-1}\sigma = 1.1496$ |
| 1 | $I_1$ | $g(\cdot; \sqrt{1.1496^2 + 1.5^2}) = g(\cdot; 1.8899)$ | $\sigma_1 = k^1\sqrt{k^2-1}\sigma = 1.4484$ |
| 2 | $I_2$ | $g(\cdot; \sqrt{1.4484^2 + 1.8899^2}) = g(\cdot; 2.3811)$ | $\sigma_2 = k^2\sqrt{k^2-1}\sigma = 1.8249$ |
| 3 | $I_3$ | $g(\cdot; \sqrt{1.8249^2 + 2.3811^2}) = g(\cdot; 3.0) = g(\cdot; 2\sigma)$ | |

we again use equation (6) to obtain

$$\begin{aligned} & k^{p+1}\sigma = \sqrt{\sigma_p^2 + (k^p\sigma)^2} \\ \Leftrightarrow \quad & k^{2p+2}\sigma^2 = \sigma_p^2 + k^{2p}\sigma^2 \\ \Leftrightarrow \quad & \sigma_p^2 = \left(k^2 - 1\right)\left(k^{2p}\sigma^2\right) \\ \Leftrightarrow \quad & \sigma_p = k^p\sqrt{k^2-1}\sigma. \end{aligned}$$

Example values for $\sigma_p$ and the incremental convolution are shown in table 2.1 for $s = 3$, $\sigma = 1.5$ and $k = 2^{\frac{1}{s}} = 2^{\frac{1}{3}}$.

**Feature localization**

Lindeberg [29] examined scale-space properties and for blob detection proposes the detection of extrema points in the space defined by equation (3) from section 2.2.2, the scale-normalized Laplacian of Gaussian $\sigma^2\nabla^2 G$ as stable feature locations. Given the above discretization of $G$, how to efficiently find those minima and maxima?

Following Lowe [33], for the general Gaussian scale-space we have the heat diffusion equation

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G. \tag{7}$$

To obtain $\nabla^2 G$ numerically, the left side of equation (7) can be approximated with a forward finite difference stencil as

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(\cdot; k\sigma) - G(\cdot; \sigma)}{k\sigma - \sigma}. \tag{8}$$

Rewriting equation (8) as Difference of Gaussian, we obtain

$$G(\cdot; k\sigma) - G(\cdot; \sigma) \approx (k-1)\sigma^2\nabla^2 G, \tag{9}$$

which shows that if the two $\sigma$ values of the two Gaussian convolved images differ by a constant factor $k$, then except for a constant factor $(k-1)$, the $\sigma^2$ scale-normalized Laplacian of Gaussian can be approximated to first order accuracy using the Difference of Gaussian filter.

Figure 2.4: Sunflower image used in the experiments.

The Difference of Gaussian approximation without the constant $(k-1)$ factor has extrema points at the same position, so in order to find these points, we proceed as follows. First, the Difference-of-Gaussian (DoG) planes $D$ are explicitly constructed from each Gaussian scale-space octave as

$$D(k^p\sigma) = G(k^{p+1}\sigma) - G(k^p\sigma) = I_{p+1} - I_p.$$

Second, the 26-neighborhood of each point – the 8-neighborhood in each plane, plus the nine adjacent top and bottom DoG planes – is searched for a maximum or minimum. The bottom-most DoG plane and the top-most DoG plane in each octave are not searched because not all 26 neighbors are known.[4] Extrema points are marked.

An example of such extrema points for the image shown in figure 2.4 is shown in figure 2.5 and 2.6. The extrema points naturally correspond to bright or dark blobs, surrounded by a dark or bright area, respectively.

Lets relate the results shown in figure 2.5 to an intuitive understanding what the Difference of Gaussian filter does. We have shown that it approximates a scale-normalized Laplacian of Gaussian filter. The Laplacian of Gaussian $LoG(\cdot;\sigma)$ is given by the following equation:

$$LoG(x,y;\sigma) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The form of the $LoG$ filter is shown in figure 2.7, and Fisher et al. [15] give an intuitive explanation what a convolution with this filter means: the filter response will have its peaks where a central region differs in intensity compared to the surrounding area. For example, consider the sunflower pictures in figure 2.5, where extrema are found where a dark area (the sunflower center) is surrounded by a bright surrounding (the leaves).

[4] In the implementation the topmost plane is convolved one more time to produce $G(\cdot;k^{s+1}\sigma)$. This allows the construction of $D(k^s\sigma)$ and hence all planes to be continuously searched.

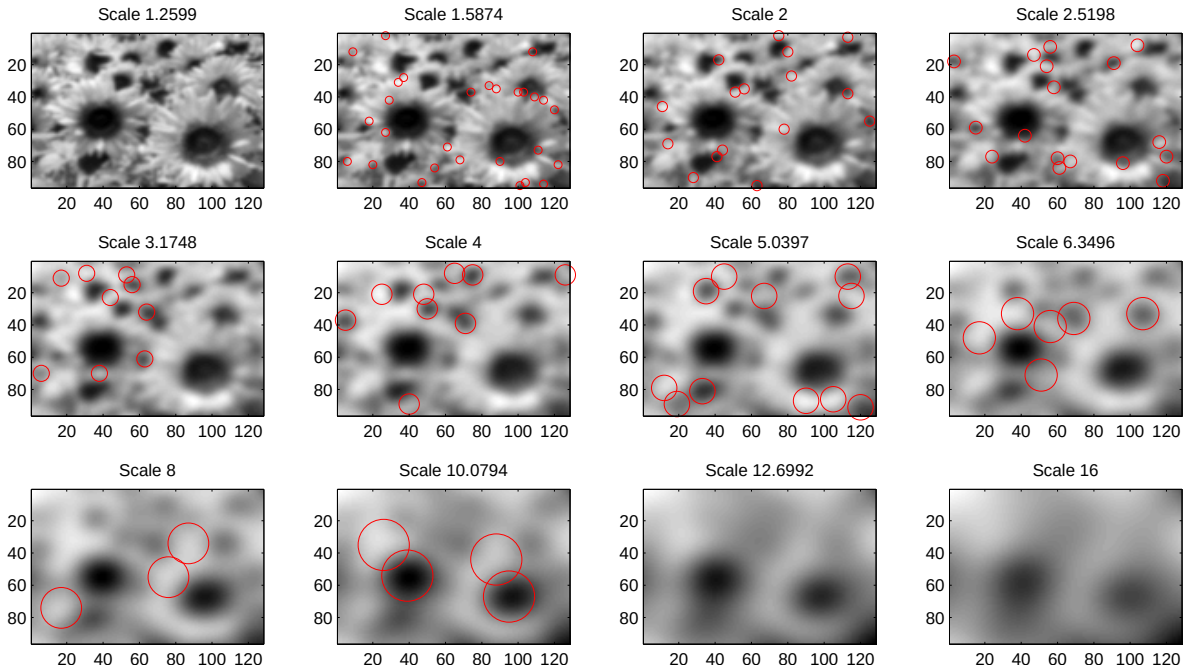


Figure 2.5: Image 2.4 in the Gaussian scale-space with SIFT extrema marked.

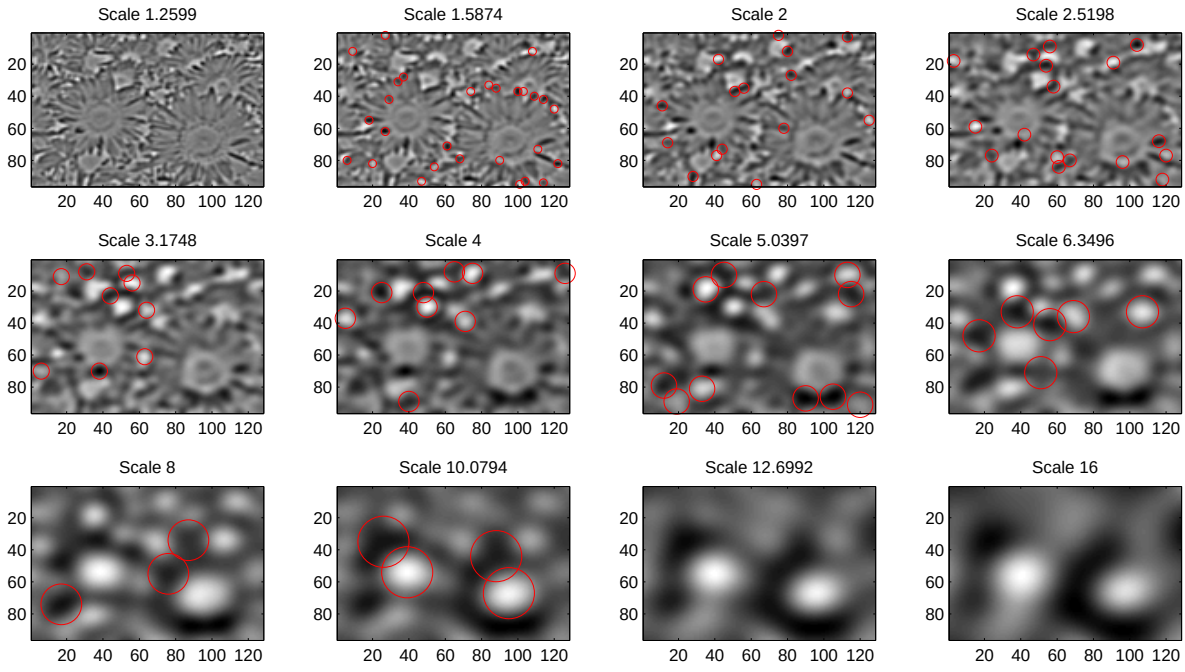


Figure 2.6: Image 2.4 in the Difference-of-Gaussian space with extrema marked.

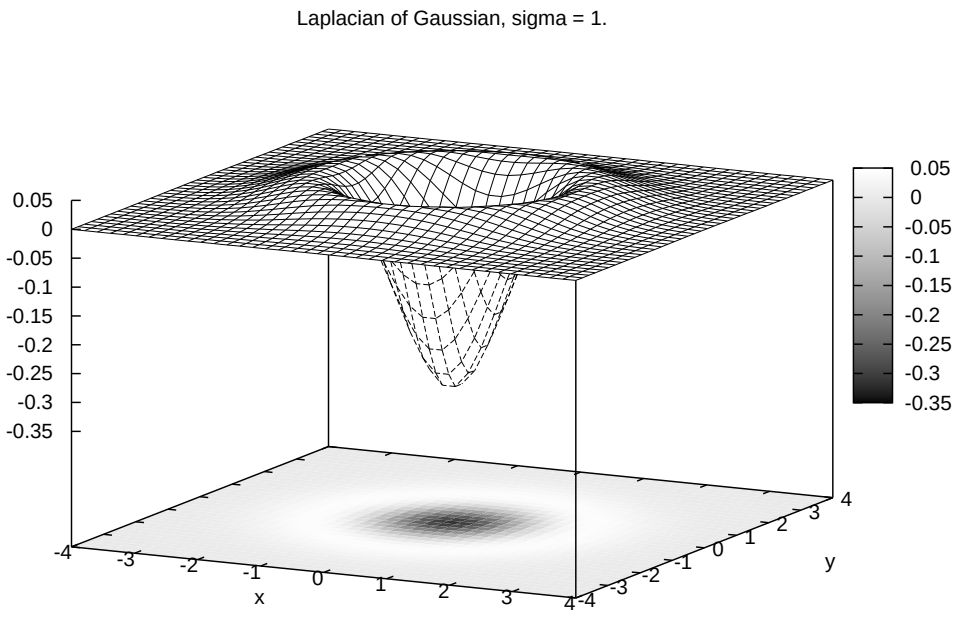


Figure 2.7: The Laplacian-of-Gaussian filter $LoG(x, y; \sigma)$ with $\sigma = 1$. Notice the basic shape: a round center with a large negative value is surrounded by a circular ring of positive values.

**Subsample accuracy.** In the original SIFT paper [32] the central point of the 26-neighborhood has been used as keypoint center. Later, Brown [5] discovered that accuracy can be significantly improved by estimating a subpixel position for the keypoint. To do this, a quadratic function is fit to the neighborhood around the maximum point and the maximum of this function calculated analytically to obtain a refined keypoint position $\hat{\mathbf{x}}$. Within the calculation, the absolute value $D(\hat{\mathbf{x}})$ is used to discard keypoints with weak extrema. The overall process is rather technical and does not add to the discussion here, therefore we omit it and point the interested reader to [33] for details.

**Keypoint pruning.** From the shape of the LoG filter shown in figure 2.7 one can see that it will produce a strong response on blobs and corners. But it will also produce a relatively strong response near straight or slightly curved edges. Using such extrema as keypoints is not desirable because the position of such an extrema is not robust against even small noise. This is due to nearby points on the same edge having similar filter response values. Which one of these values then becomes the extrema depends only on small noise on the edge.

To prune such keypoints, Lowe [33] proposes to use the idea of the Harris corner and edge detector [22] to eliminate points that have a large principal curvature only in one spatial direction but not in the perpendicular one. The principal curvature is not computed explicitly but the ratio $r$ of the larger to the smaller principal curvature is computed and keypoints

with a large ratio are pruned. Lowe derives the computation by the Hessian

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix},$$

where the derivatives are approximated using finite differences. He finally obtains the equation

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(D_{xx} + D_{yy})^2}{D_{xx}D_{yy} - (D_{xy})^2} = \frac{(r+1)^2}{r}. \tag{10}$$

Given a value of $r$ all keypoints above the quotient $\frac{(r+1)^2}{r}$ can be removed by computing the left hand side of equation (10).[5]

Summarizing, we now have a set of keypoints for an image, and each keypoint has a subsample accurate position in scale-space. The points have been filtered such that only points with a large enough LoG filter response remain and such that their ratio of principal curvatures are below a given threshold. What remains is to construct one feature descriptor per point from their neighborhood.

Example Matlab code producing a Difference-of-Gaussian space like the one used in SIFT, doing keypoint localization and pruning is given in section A.1.

### Construction of an invariant feature descriptor

For each keypoint, an *orientation* is explicitly assigned and all following operations are relative to this orientation. This realizes *rotation invariance* of the keypoint descriptor. To assign the orientation, the following steps are done.

1. A 36 bin histogram storing gradient directions is initialized to all zero.
2. A circular weighting function is anchored to the keypoint, here a Gaussian with $\sigma = 3\rho$ is used, where $\rho$ is the scale the keypoint was detected at.
3. For each pixel in reach of the Gaussian[6] the gradient magnitude $m$ of the pixel is distributed linearly interpolated over one or two bins chosen by the gradient orientation $\theta$. $m$ and $\theta$ are precomputed for each pixel in the Gaussian scale-space in the plane $G$ closest to the given $\rho$. Here $m$ and $\theta$ are calculated in the standard finite difference fashion:

$$m = \sqrt{(G(x+1,y) - G(x-1,y))^2 + (G(x,y+1) - G(x,y-1))^2},$$

$$\theta = \tan^{-1}\left(\frac{G(x,y+1) - G(x,y-1)}{G(x+1,y) - G(x-1,y)}\right).$$

   The contribution of each gradient magnitude to the histogram bin is weighted by the Gaussian, such that points far away from the point receive a small weight.
4. The histogram is examined for peak bins and a keypoint descriptor is created for any orientatin, whose corresponding peak value is within 80% of the highest valued bin.

   This means a single keypoint can produce more than one SIFT descriptor, but this contributes to the stability.

[5] Lowe recommends $r = 10$.

[6] Normally a sampling window with a length of $3\sigma$ in each dimension is used.

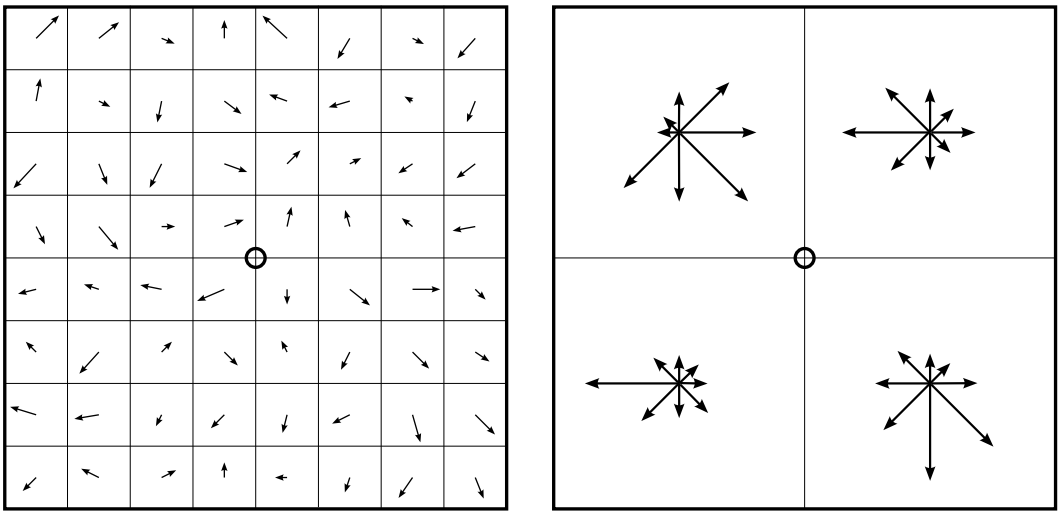

Figure 2.8: Creating a SIFT descriptor from an anchored sampling window (left): the gradient magnitudes are sampled at each point from the sampling window into four gradient histograms (right). The position of the pixel in the anchoring window determines the distribution among the histograms. The distribution of the partial magnitude into the bins of each histogram is determined by the gradient orientation. Here four histograms with eight bins each are used, while in the real SIFT descriptor 4x4 histograms of 16 bins are used, each bin occupying an angular part of 22.5° degrees. The sampling window is additionally weighted by a circular Gaussian.

After the orientation of each keypoint has been established, the actual descriptor is constructed by anchoring a sampling window around the keypoint in the given orientation and sampling the gradients in a multiple gradient histogram. The basic idea is shown in figure 2.8, which on the left side shows the sampling window with its center marked and on the right side shows the grouped gradient histograms.

The square sampling window is anchored by the position, orientation and scale of the keypoint: the position of the keypoint is the center point, the orientation determines the $y$-axis of the window and the size of the window is determined by the keypoints scale.[7]

Over this reference sample window a Gaussian weighting is centered with a $\sigma$ of half the descriptor window length in each dimension. For every pixel in the window the gradient magnitude is distributed in the gradient histograms, as explained in the caption of figure 2.8. As all such assignments are done using linear interpolation between the bins and the histograms, the descriptor is quite robust against slight shifts. Additionally, the Gaussian weighting adds robustness against changes far away from the keypoint center.

The histograms are normalized and stored linearly as *SIFT descriptor*, which for the real SIFT has 128 elements.

**Summary.** Some details have been omitted from the description of the SIFT algorithm as they do not add much to the discussion of the basic principles. The interested reader is

[7] Lowe [33] does not give a window size for the sampling window but it is known that in his demo program he uses a $9\rho$x$9\rho$ sampling window, where $\rho$ is the keypoints scale. The authors own implementation (libsift) uses $8\sqrt{2}\rho$ in each dimension.

invited to review the author's SIFT implementation source code.[8]

Although the original SIFT algorithm was proposed in 1999, the small improvements it has received over time still make it the first choice as robust local image feature extraction method. It is efficient to compute, robust and well understood. There have been some attempts to improve upon the original SIFT in a number of ways. First, the problems of high dimensionality of the original SIFT descriptors have been addressed with PCA-SIFT [26], which reduces the 128 dimensions of the original SIFT descriptor to 40 by principal components analysis. Second, in BSIFT [50], the descriptor can be adjusted using a known object segmentation mask so it only incorporates information from the foreground. Third, although most often the bottleneck of a computer vision system is located after the SIFT feature extraction, Grabner et. al [19] replaced the DoG approximation to the LoG filter with a Difference-of-Boxes (DoB) filter [15] computable efficiently by an integral image yielding a speedup factor of eight.

The other variants of SIFT proposed in the literature so far are of little interest as they do not generally outperform SIFT in object recognition tasks. A detailed comparison of local feature detectors is [39], a comparison of feature descriptors is [38].

## 2.3 Support Vector Machines

Support Vector Machines (SVM) are an effective class of machine learning algorithms developed in the 1990'ies. SVMs and the more general *kernel methods* are a very active area of research and in this section we limit our description to the types of SVMs by the extend they are used in this thesis. For a comprehensive introduction into kernel methods, see Schölkopf and Smola [46]. For a general overview of pattern classification systems, see Duda et al. [12].

We first give the theoretical motivation behind Support Vector Machines, followed by the simplest possible SVM, the hard margin linear SVM. The larger remaining part of the section then deals with three important generalizations, namely i) the extension to the non-linear case, ii) the non-separable case and iii) multiclass classification SVMs. Additionally, in a fourth subsection we discuss how to overcome the problem of large diagonal elements in a kernel matrix, which is a common problem in kernels for highly structured data.

### 2.3.1 Pattern Recognition Learning Machines

We now give a short introduction into pattern recognition learning machines to setup the context for the discussion of Support Vector Machines in in the following sections. We mainly follow the excellent tutorial on SVMs by Burges [6], giving more recent results where possible.

#### Statistical learning theory

Support Vector Machines were developed as a result of statistical learning theory [53]. Consider a system that learns from given training data. Generally, one is interested in designing the system to *generalize* well, meaning the trained system will be able to give correct output even for previously unseen data. One observation that the statistical learning theory makes about all such systems is that having a good generalization ability is a balance between the ability to learn the *given training set* and the ability to learn *any given set*. The later is the

[8] Available at `http://cs.tu-berlin.de/~nowozin/libsift/`

"*capacity*" of the learning system. Now consider the following three cases for the capacity of a learning system.

- Overly high capacity.

  The high capacity of the learning system allows it to learn any set of training patterns correctly. Thus, it captures the characteristics of the given training set well, but does not capture the real structure of the data. This is known as *overfitting*, and the resulting system will not perform well on new samples.

- Overly low capacity.

  The low capacity does not allow the system to learn the given training set well. If not even the training set is well learned, the structure of the data is not captured and the system is unlikely to perform well on new data.

- "Right" degree of capacity.

  The capacity is high enough to allow the system to learn the given training data well. Yet the capacity is low enough to not overfit on the training data, therefore the real structure of the data is captured by the trained system and it will perform well on new data.

Now let us put the above ideas in a more formal and practical way.

### Bounds on Generalization Performance

Let $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, \ldots, \ell$ be some observations, and $y_i \in \{-1, 1\}$ be labels of $\mathbf{x}_i$ to one of two classes. We assume the labels $y_i$ to be correct without doubt, that is, we deal only with perfect training data.

Although we limit $y_i$ to a fixed assignment, consider the more general case where $P(\mathbf{x}, y)$ is a probability distribution from which the data is drawn *independently and identically distributed* (iid). In this case, the labels are not absolute for a given point $\mathbf{x}$.

Now further let a learning machine be a family of functions

$$f(\mathbf{x}, \alpha),$$

parametrized on a set of adjustable parameters $\alpha$, providing a label prediction for the observation $\mathbf{x}$. Training such a learning machine is the task of finding a suitable set of parameters $\alpha$ so that $f(\mathbf{x}, \alpha)$ matches the known labels $y_i$ for many training samples. Clearly, a function family with sufficiently high capacity will always allow one to find an $\alpha$ so that all fixed labels of a given training set are correctly predicted.

The generalization ability of such a trained system is important to us, and is measured by the expected error on the test samples drawn from the same probability distribution $P$. The expected test error of the system, also called the "actual risk", is given by

$$R(\alpha) := \int \frac{1}{2} \left| y - f(\mathbf{x}, \alpha) \right| dP(\mathbf{x}, y), \tag{11}$$

where $\frac{1}{2} \left| y - f(\mathbf{x}, \alpha) \right|$ is called the *loss*. For a single pair $(\mathbf{x_i}, y_i)$ it can take the values zero or one.

Unfortunately, the actual risk cannot be computed, as we usually have no information about $P(\mathbf{x}, y)$. If we would, we could pick the optimal set of parameters $\alpha$, for which $R(\alpha)$ is minimized. Unlike $R(\alpha)$, we can compute the *empirical risk* $R_{emp}(\alpha)$, which is defined as the mean error rate on the training set, hence we have

$$R_{emp}(\alpha) := \frac{1}{l} \sum_{i=1}^{\ell} \frac{1}{2} |y_i - f(\mathbf{x}_i, \alpha)| . \tag{12}$$

We are now in a position to use the empirical risk $R_{emp}(\alpha)$ and a notion of capacity of the learning system - the Vapnik-Chervonenkis dimension (VC-dimension) - to provide a bound on the actual risk $R(\alpha)$. With a chosen probability $\eta$, $0 \leq \eta \leq 1$, the following bound holds[9]:

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h\left(\log\left(\frac{2\ell}{h}\right) + 1\right) - \log\left(\frac{\eta}{4}\right)}{\ell}}, \tag{13}$$

where $h$ is the VC-dimension of the system, which we explain below. Given knowledge of $h$ we can compute the right hand side of the bound without requiring knowledge about $P(\mathbf{x}, y)$.

#### Vapnik-Chervonenkis dimension

Consider the same setup as before, with $\ell$ given points $\mathbf{x}_i$. Now further consider all possible $2^\ell$ labellings for these points. A function family $f(\mathbf{x}, \alpha)$ can *shatter* the given set of points, if for all possible labellings an $\alpha$ can be found, so that $f(\mathbf{x}, \alpha)$ correctly assigns all labels to the points. The VC-dimension $h$ of $f(\mathbf{x}, \alpha)$ is the maximum number of points for which $f(\mathbf{x}, \alpha)$ can shatter the set.

As an example, lets take as function the set of all possible hyperplanes in $\mathbb{R}^n$. Then the VC-dimension is $n + 1$. For $n = 2$, there exist at least one set of three points that can for all possible labellings be separated by the hyperplane. However, no set of four points can be separated for all possible labellings. This is illustrated in figures 2.9 and 2.10.

As Burges [6] notes, there is no relationship between the number of parameters in $\alpha$ and the VC-dimension, as one would intuitively expect.

Also note, that there can be learning systems with an infinite VC-dimension. For those systems, the bound (13) is not applicable. But nevertheless, systems with such infinite capacity can perform very well.

### 2.3.2 Hard margin SVM

We now give a definition of the hard margin support vector machine followed by an intuitive explanation how it works. We mainly follow Burges [6].

Let $\{(\mathbf{x}_i, y_i)\}_i$, $i = 1, \ldots, \ell$ be a set of sample points $\mathbf{x}_i \in \mathbb{R}^d$ with associated class labels $y_i \in \{-1, 1\}$. The set is said to be *separable* if a hyperplane exists which divides $\mathbb{R}^d$ such that each halfspace only contains the points of one label, -1 or 1 respectively. Any hyperplane living in $\mathbb{R}^d$ can be described by $(\mathbf{w}, b)$, where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector normal to the plane and $b \in \mathbb{R}$ is a bias. The hyperplane consists of all points $x \in \mathbb{R}^d$ for which

$$\mathbf{x} \cdot \mathbf{w} + b = 0. \tag{14}$$

[9] Proof in Vapnik [53].

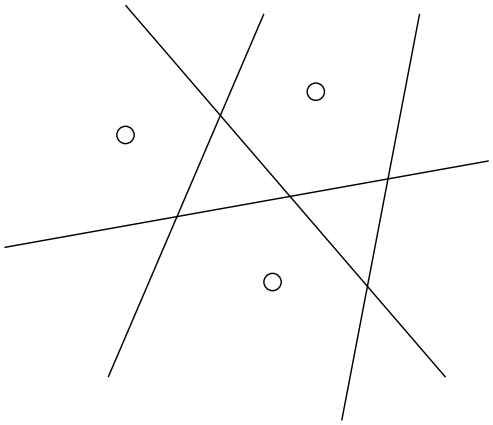

Figure 2.9: This set of three points in $\mathbb{R}^2$ can be separated for any possible labeling by oriented hyperplanes in $\mathbb{R}^2$, as shown.

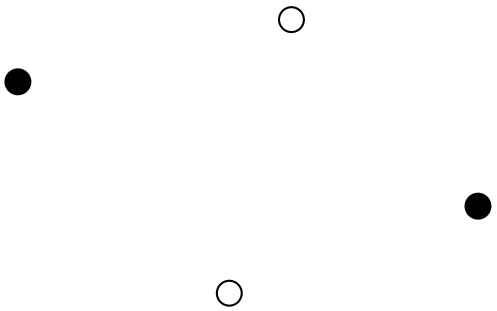

Figure 2.10: There is no set of four points in $\mathbb{R}^2$ that can be separated for *all* possible labellings. This is an example set which illustrates a "difficult" labeling.

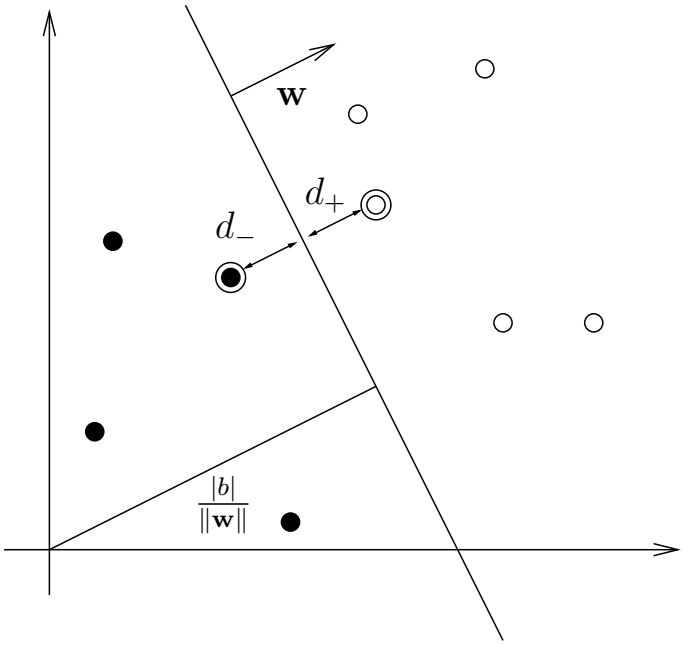


Figure 2.11: Basic support vector machine setup: two sets of labeled points, separated by a hyperplane. The hyperplane is parametrized by $(\mathbf{w}, b)$, where the distance to the origin is given by $\frac{|b|}{\|\mathbf{w}\|}$. The points closest to the hyperplane (circled) are called *support vectors* and their distance to the hyperplane is $d_+$ and $d_-$, respectively. The sum $d_+ + d_-$ is the margin width.

Points $\mathbf{x} \in \mathbb{R}^d$ not on the hyperplane have a signed distance of $\frac{\mathbf{x}\cdot\mathbf{w}+b}{\|\mathbf{w}\|}$ to the hyperplane,

where the sign gives the index of the halfspace the point is contained in.

Let $d_+$ ($d_-$) be the smallest distance of any positive (negative) sample point to the hyperplane. The *margin* is the sum $d_+ + d_-$ and gives the width of the band parallel along the hyperplane that does not contain any sample points. Naturally, a large margin separates the sample points more clearly than a small one. This idea leads to the optimization goal of a Support Vector Machine: to separate the samples with the largest possible margin.

If there is a margin bigger than zero, we can - by scaling $\|w\|$ - formulate the following constraints:

$$\begin{aligned} \mathbf{x}_i \cdot \mathbf{w} + b &\geq 1, && \forall i \text{ where } y_i = 1, \\ \mathbf{x}_i \cdot \mathbf{w} + b &\leq -1, && \forall i \text{ where } y_i = -1. \end{aligned}$$

Using the definition of $y_i$, we combine these two constraints into the constraint

$$y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) - 1 \geq 0, \qquad \forall i. \tag{15}$$

Consider the points closest to the hyperplane, with a distance $d_+$ or $d_-$, respectively. These points fulfill the exact equality of equation (15), and thus $\mathbf{x} \cdot \mathbf{w} + b = 1$ for them. It follows that $d_+ = d_- = \frac{1}{\|\mathbf{w}\|}$ and the margin is $d_+ + d_- = \frac{2}{\|\mathbf{w}\|}$. Now, in turn of maximizing the margin $d_+ + d_-$ we can instead minimize $\|\mathbf{w}\|^2$, subject to constraint (15). Obtaining the hard margin SVM solution is then equivalent to solving the optimization problem

$$\begin{aligned} &\min_\alpha \|\mathbf{w}\|^2, \\ &\text{sb.t. } y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) - 1 \geq 0, \qquad \forall i. \end{aligned} \tag{16}$$

The points whose distance to the hyperplane is $d_+(d_-)$ are called *support vectors*, as the position of the hyperplane is determined by them exclusively. In fact, any other point – the *non support vectors* – can be removed without moving the hyperplane.

### 2.3.3 Lagrangian formulation

So far we have considered the hard margin SVM as optimization problem. To solve this constrained optimization problem we have to reformulate (16) in what is called the *Lagrangian formulation*. A modern introduction into optimization is by Pedregal [41], a more theoretical account from Sawaragi et. al [43].

The Lagrangian form of (16) has three important properties:

- the problem has the form of a standard constrained *quadratic programming* problem, for which solvers are readily available,
- the constraints (15) are easier to handle,
- in the final formulation we arrive at, we access the training data only by means of the dot-product.[10]

We now show how the optimization can be brought to the Lagrangian form. This is a standard approach in solving constrained optimization problems.

[10]This, as we will see, will be a very important requirement for extending the linear SVM to the non-linear case.

We introduce positive Lagrange multipliers $\alpha_i$, $i = 1, \ldots, \ell$ for each constraint (15). The constraint (15) can be written for every sample $i$ as $c_i = y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1$ and as we have $c_i \geq 0$, we obtain the primal form:

$$L_P = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \alpha_i c_i = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{\ell} \alpha_i \tag{17}$$

$L_P$ in equation (17) is minimized with respect to $\mathbf{w}$, $b$ and under the requirement that the derivatives of $L_P$ with respect to all $\alpha_i$ vanish. This problem is convex [6], and because of its form a solution can be obtained by solving the corresponding *dual problem*. This is done by maximizing $L_P$, subject to the constraint that the gradient of $L_P$ with respect to $\mathbf{w}$ and $b$ vanishes. Equivalent means the maximum of the dual problem is obtained at the same choice of $\alpha_i$ as the minimum of the primal problem.

Requiring that the gradient of $L_P$ with respect to $\mathbf{w}$ and $b$ vanishes gives the conditions

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i \tag{18}$$

and

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0 \tag{19}$$

as exact *equality constraints*. As we have equality in (18) and (19), we can resubstitute them back into equation (17) to obtain the *dual problem* as follows

$$L_D = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{20}$$

Training an SVM can then be cast into the following optimization problem

$$\begin{aligned} & \max_{\alpha} L_D \\ \text{sb.t.} \quad & \textstyle\sum_{i=1}^{\ell} \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \forall i = 1, \ldots, \ell. \end{aligned} \tag{21}$$

$L_P$ and $L_D$ have the same solution, however it is obtained from two different formulations and two different sets of constraints. Training a SVM resolves to solving problem (21), resulting in a set of Lagrange multipliers $\alpha_i$, one multiplier for every sample $\mathbf{x}_i$. If $\alpha_i > 0$ is obtained, $\mathbf{x}_i$ is a support vector. The parameters $(\mathbf{w}, b)$ of the hyperplane are readily obtained given $\alpha_i$ by using equation (18) and by solving the exact equality for support vectors in equation (15), obtaining $b$ as

$$b = y_i - \mathbf{x}_i \cdot \mathbf{w} = y_i - \mathbf{x}_i \cdot \sum_{j=1}^{\ell} \alpha_j y_j \mathbf{x}_j$$

using any support vector $\mathbf{x}_i$ for which $\alpha_i > 0$.

The optimization problem for Support Vector Machines is *convex*, having one unique solution. This contrasts with other machine learning approaches, such as neural networks,

which often have many local extrema. Solving the dual formulation has a beautiful geometric interpretation: it corresponds to minimizing the distance between two points on the convex hull of the two sets [6, 1]. This can be used to obtain fast approximate online methods which converge to the real SVM solution, as done by Bordes and Bottou [2].

In the next sections we will see how the linear hard margin SVM can be extended in two ways, namely

- by introducing non-linearity, allowing nonlinear decision surfaces between the training samples to be learned, and
- by gracefully handling the non-separable case, which is of high importance for solving real world data sets, which are often not separable.

**Example**

A small example implementation of the dual form linear hard margin SVM (21) is shown in section A.1. The reader is encouraged to try it and plot the resulting hyperplane to verify the correctness.

### 2.3.4 Kernels

So far, we have seen how hard-margin SVMs can separate points by means of a linear hyperplane. In the dual Lagrangian formulation, the training samples are accessed only through the dot product between two samples. This property allows a beautiful extension of the linear case to the non-linear one, simply by replacing every dot product with a so called *kernel function*, or *kernel* for short.

We will examine what requirements are imposed on kernels to be "legal" kernels and how to construct kernel functions, but first, to better understand what a kernel is, consider the following example [46]. We have training samples $\mathbf{x}_i \in \mathbb{R}^2$. Let $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ be a mapping of the input space defined as

$$\phi(\mathbf{x}) := \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}.$$

Now consider $\langle \phi(\mathbf{x}_A), \phi(\mathbf{x}_B) \rangle$, the dot product between two mapped samples:

$$\begin{aligned} \langle \phi(\mathbf{x}_A), \phi(\mathbf{x}_B) \rangle &= \left\langle \begin{bmatrix} x_{A_1}^2 \\ x_{A_2}^2 \\ \sqrt{2}x_{A_1}x_{A_2} \end{bmatrix}, \begin{bmatrix} x_{B_1}^2 \\ x_{B_2}^2 \\ \sqrt{2}x_{B_1}x_{B_2} \end{bmatrix} \right\rangle \\ &= x_{A_1}^2 x_{B_1}^2 + x_{A_2}^2 x_{B_2}^2 + 2x_{A_1}x_{A_2}x_{B_1}x_{B_2} \\ &= (x_{A_1}x_{B_1} + x_{A_2}x_{B_2})^2 = \langle \mathbf{x}_A, \mathbf{x}_B \rangle^2. \end{aligned}$$

The last equality is quite surprising: it states that to compute $\langle \phi(\mathbf{x}_A), \phi(\mathbf{x}_B) \rangle$, we do not actually have to compute the mapping $\phi(\mathbf{x})$, but instead can still use only the original samples $\mathbf{x}_A$, $\mathbf{x}_B$. In our case, we explicitly constructed $\phi(\mathbf{x})$ and then evaluate a dot product on elements of the new mapped space, so we know it is a valid dot product and can be used in our SVM. But what does the use of a mapping $\phi$ change?

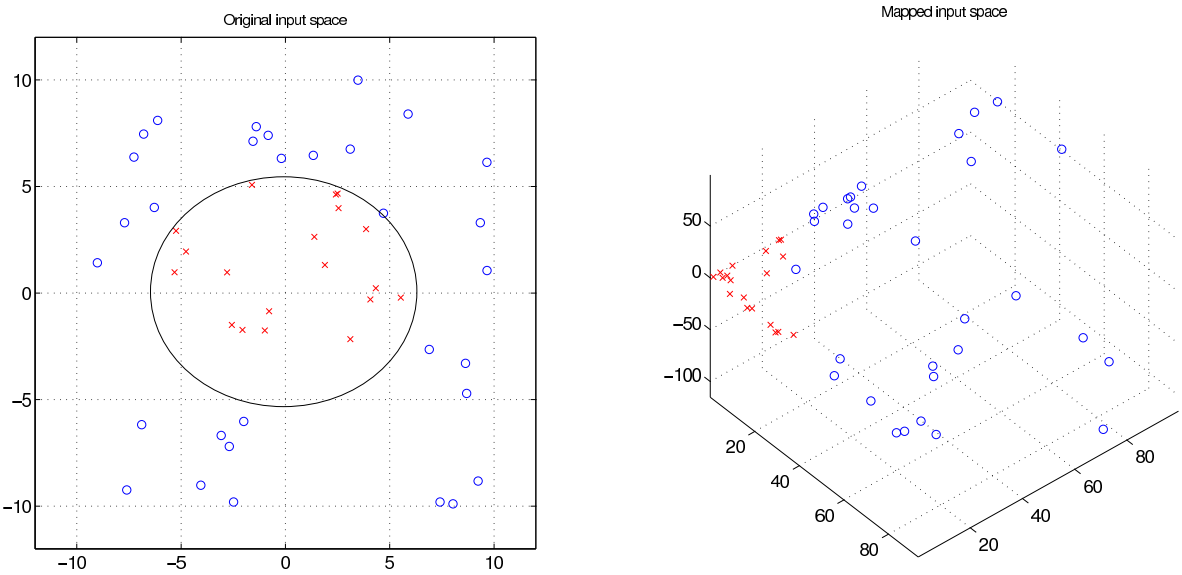


Figure 2.12: Mapping an input space $\mathbb{R}^2$ to a higher dimensional space $\mathbb{R}^3$ can make the data linearly separable.

### Example

Consider the problem of learning a decision function from the two class samples in $\mathbb{R}^2$ shown in the left part of figure 2.12. Obviously, a linear SVM cannot separate the two classes successfully. However, by first mapping the data into a higher dimensional space by means of $\phi$, we obtain a new set of mapped training samples which can be linearly separated. This is shown on the right side of figure 2.12, where $\phi$ is the mapping defined above.

If we map the decision boundary defined by the separating hyperplane obtained in the high dimensional space back into the original input space, we yield a non-linear decision boundary, in this case an ellipse. By plugging in the dot product $\langle \phi(\cdot), \phi(\cdot) \rangle$ into our original SVM formulation (20), we extended the linear SVM to the non-linear case.

### Definition

Let us formalize the notion of "kernel" now. Let a *kernel function* $K(\mathbf{x}_A, \mathbf{x}_B)$ be defined as

$$K(\mathbf{x}_A, \mathbf{x}_B) := \langle \phi(\mathbf{x}_A), \phi(\mathbf{x}_B) \rangle,$$

where $\phi(\cdot)$ is a mapping of the space the samples $\mathbf{x}$ live in, into a *Hilbert space* $\mathcal{H}$ with a defined dot product.

Note that this definition is not really giving us an idea how $\mathcal{H}$ looks like or how to construct a kernel function $K$. In fact, we often only know that at least one such $\mathcal{H}$ exists and hence a function is a kernel function. Given only a function $K(\cdot, \cdot)$, the requirements for $K$ to be a legal kernel function are given by *Mercer's condition* [6]:

There exist a mapping $\phi$ and an expansion

$$K(\mathbf{x}_A, \mathbf{x}_B) = \sum_i \phi(\mathbf{x}_A)_i \phi(\mathbf{x}_B)_i$$

iff for any $g(\mathbf{x})$ such that

$$\int g(\mathbf{x})^2 dx$$

is finite, then

$$\int K(\mathbf{x}_A, \mathbf{x}_B) g(\mathbf{x}_A) g(\mathbf{x}_B) d\mathbf{x}_A d\mathbf{x}_B \geq 0.$$

It might be difficult to check Mercer's condition for some cases, but we now give a brief list of commonly used kernel functions known to be valid.

### Commonly used kernel functions

The following kernel functions are popular for support vector machine learning where the input domain is $\mathbb{R}^n$.

- "Linear" kernel.

  $$K(\mathbf{x}_A, \mathbf{x}_B) := \langle \mathbf{x}_A, \mathbf{x}_B \rangle$$

  is the simplest kernel function which uses the identity mapping $\phi(\mathbf{x}) = x$ and produces linear classifiers when used with SVMs.

- (Homogeneous) polynomial kernel.

  $$K(\mathbf{x}_A, \mathbf{x}_B) := \langle \mathbf{x}_A, \mathbf{x}_B \rangle^d,$$

  where $d$ is the degree of the polynomial. The corresponding Hilbert space is known to have $\binom{n+d-1}{d}$ dimensions.

- Inhomogeneous polynomial kernel.

  $$K(\mathbf{x}_A, \mathbf{x}_B) := (\langle \mathbf{x}_A, \mathbf{x}_B \rangle + c)^d,$$

  $d \in \mathbb{N}$, $c \in \mathbb{R}^+$.

Often, *invariance* in the input space against translation and rotation are desired. Then, *radial basis function* kernels (RBF kernels) can be used, which all have the general form $K(\mathbf{x}_A, \mathbf{x}_B) = f(d(\mathbf{x}_A, \mathbf{x}_B))$, where $d(\cdot, \cdot)$ is a metric in the input space and $f : \mathbb{R}^+ \to \mathbb{R}^+$ is a positive function. The most popular general RBF kernel for support vector machines is the Gaussian RBF kernel.

- Gaussian RBF kernel.

  $$K(\mathbf{x}_A, \mathbf{x}_B) := e^{-\frac{\|\mathbf{x}_A - \mathbf{x}_B\|^2}{2\sigma^2}},$$

  where $\sigma$ is the standard deviation. Relating the Gaussian RBF kernel to the general RBF form, we have $d(\mathbf{x}_A, \mathbf{x}_B) = \|\mathbf{x}_A - \mathbf{x}_B\|$ and $f(\cdot) = e^{-\frac{(\cdot)^2}{2\sigma^2}}$.

Intuitively, a kernel function can be related to a distance metric in the following way[11]: a distance metric measures dissimilarity, having large values where samples differ, but a kernel measures similarity, having a value of zero when samples differ and a large value when they a similar.

The listed kernel functions all require $\mathbb{R}^n$ as input space, but as we will see, in general this is not required. There are kernels for non vectorial data as well as structured data. One such kernel, the Marginalized Graph Kernel, will be examined in detail in section 2.4.

Practically, for small training sets ($\ell < 10^4$), the *kernel matrix*, $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ can be used to express all information about the samples that is available to the learning method. For larger sets, the kernel matrix is not explicitly stored, but the kernel values are calculated and cached when necessary.

### Constructing kernels

There are many ways to construct new kernel functions. We will limit ourselves to only two methods of composing kernel functions that will be useful later.

1. Weighted sums of kernel functions.

   If $K_1$, $K_2$, ..., $K_n$ are valid kernel functions, then for $\alpha_i \in \mathbb{R}^+, i = 1, \ldots, n$, the function

   $$K(\mathbf{x}_A, \mathbf{x}_B) := \sum_{i=1}^{n} \alpha_i K_i(\mathbf{x}_A, \mathbf{x}_B)$$

   is a valid kernel.

2. Product of kernel functions.

   If $K_1$, $K_2$, ..., $K_n$ are valid kernel functions, then the product kernel

   $$K(\mathbf{x}_A, \mathbf{x}_B) := \prod_{i=1}^{n} K_i(\mathbf{x}_A, \mathbf{x}_B)$$

   is a valid kernel.

A comprehensive description on how to construct kernels is given by Schölkopf and Smola [46].

### Norm and distances

Even though the mapping $\phi(\mathbf{x})$ is implicit in the kernel function, we can evaluate some important properties of the mapped features, such as the norm of a mapped feature and the distance between two mapped features [47].

**Norm.** The norm can be expressed as

$$\|\phi(\mathbf{x})\|_2 = \sqrt{\|\phi(\mathbf{x})\|^2} = \sqrt{\langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle} = \sqrt{K(\mathbf{x}, \mathbf{x})}.$$

[11] Although such kind of comparison always breaks down quickly when you come to rely on it, it is helpful to get a clearer idea of what makes up a "good" kernel: samples that share the property we are concerned with (say, their class membership) should yield a high kernel value when compared with each other and a low value when compared with other samples. Thus, designing a good kernel requires balancing discriminative power with similarity between like samples. A formal definition of "good kernel" can be found in Gärtner [16].

**Distance between $\phi(\mathbf{x})$ and $\phi(\mathbf{z})$.** The squared euclidean distance between two mapped feature vectors can be calculated as

$$\begin{aligned}\|\phi(\mathbf{x}) - \phi(\mathbf{z})\|^2 &= \langle \phi(\mathbf{x}) - \phi(\mathbf{z}), \phi(\mathbf{x}) - \phi(\mathbf{z}) \rangle \\ &= \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - 2\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle + \langle \phi(\mathbf{z}), \phi(\mathbf{z}) \rangle \\ &= K(\mathbf{x}, \mathbf{x}) - 2K(\mathbf{x}, \mathbf{z}) + K(\mathbf{z}, \mathbf{z}).\end{aligned}$$

### 2.3.5 Soft margin SVM

Until now we only considered the separable case, where a hyperplane can separate the data. We have used kernel functions to map the data into a higher dimensional space. But even in this space it is possible – yes, even likely for real data – that the hyperplane cannot separate the samples. For such non-separable data, the hard margin SVM (21) will not have a solution.

In case no separable hyperplane exists, for any hyperplane in feature space, there will be at least one sample that does not fulfill the constraint (15). By relaxing this constraint to allow such errors to be made we can obtain a SVM formulation that always has a solution, even for non-separable data. By additionally adding the minimization of the total sum of errors to be made into the objective function, we guarantee that errors are only made when necessary. That is, the original hard margin SVM is a recovered for separable data.

Constraint (15) is extended through the use of positive *slack variables* $\xi_i$, $\xi_i \geq 0$, $i = 1, \ldots, \ell$ adjusting for the error being made. The new constraint then becomes

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0. \tag{22}$$

If an error occurs and a sample $\mathbf{x}_j$ cannot be classified correctly with the chosen hyperplane, then $\xi_j > 1$. We chose the hyperplane as usual, but additional minimize $C\left(\sum_{i=1}^{\ell} \xi_i\right)^k$ by adding it to the objective function. $C$ and $k$ are two additional parameters, $k$ being the power of the total summed error and $C$ is a parameter controlling the penalty for errors. For $k = 1$ we conveniently obtain $L_D$ in exactly the same form as (20). $C$ has to be chosen a-priori and common values are 10, 1000 or positive infinity. In the optimization problem (21), the only change is an additional constraint on the Lagrange multipliers, so the new dual optimization problem becomes

$$\begin{aligned} & \max_\alpha L_D \\ \text{sb.t.} \quad & \textstyle\sum_{i=1}^{\ell} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \forall i = 1, \ldots, \ell. \end{aligned} \tag{23}$$

Details on this and similar soft margin formulations can be found in Schölkopf and Smola [46].

### 2.3.6 Multiclass SVM

In this section we consider the question of how to extend the methods described so far to classification problems involving more than two classes. Although there exist multiclass SVM formulations that incorporate additional terms based on the multiple classes into the objective function, we will only consider two simple voting based schemes. In practice, the classification accuracy is nearly the same; a detailed analysis is Weston and Watkins [56].

In the following discussion we assume $k$ to be the number of classes, where each training sample is member of exactly one class.

### One-vs-rest classification

The one-vs-rest classification scheme splits the training set into two classes, one containing samples in the currently considered class, the other holding the samples of all other classes. By splitting it $k$ times, each time considering a different class, the scheme obtains $k$ "one-vs-rest" classifiers. Algorithmically, the scheme works as follows.

1. Divide training set $k$ times into two-class sets:
   - The positive set: samples of the currently considered class.
   - The negative set: samples of all the other classes.
2. Train $k$ one-vs-rest classifiers.
3. Test new samples with a voting scheme between the $k$ classifiers (not specified).

### One-vs-one classification

The idea of the one-vs-one classifier is similar to the one-vs-rest decomposition. However, all possible combinations between $k$ classes, $\frac{k(k-1)}{2}$ in total are used to train the same number of classifiers, each seeing only the samples of two classes. The steps are similar to the one-vs-rest classifier.

1. Create all $\frac{k(k-1)}{2}$ possible pairings of classes.
2. Train one two-class classifier for each pairing.
3. Test new samples with a voting scheme.

The voting scheme we will use is the one implemented in the Spider machine learning toolkit and works as described in algorithm *OneVsOneVoting*. The idea is to first attempt to use a hard binary classification voting and if that fails, to use a smooth function always leading to a unique classification result.

**Algorithm** *OneVsOneVoting*
**Input:** The number of classes, $k$.
**Input:** $\frac{k(k-1)}{2}$ outputs $y_{i,j}$ of one-vs-one SVM between class $i$ and $j$ for a single sample $\mathbf{x}$.
**Output:** A class label $y$ for the sample $\mathbf{x}$.

```
1.  ScoresBinary ←∅
2.  ScoresSmoothed ←∅
3.  for i = 1 to k
4.          for j = i to k
5.                  ScoresBinary[i] ←ScoresBinary[i] + sign(y_{i,j})
6.                  ScoresBinary[j] ←ScoresBinary[j] − sign(y_{i,j})
7.                  ScoresSmoothed[i] ←ScoresSmoothed[i] + tanh(y_{i,j})
8.                  ScoresSmoothed[j] ←ScoresSmoothed[j] − tanh(y_{i,j})
9.  if there is a unique maximum in ScoresBinary at p
10.         y ←p
11.   else
12.         y ←index of maximum in ScoresSmoothed
```

### 2.3.7 Large diagonals

A kernel matrix $K$ with $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ contains all the information required by a kernel based machine learning algorithm: all kernel values between all possible pairings of the training samples.

Kernels for structured data, such as kernels on strings and graphs, can produce a kernel matrix where the diagonal values $K_{i,i}$ are orders of magnitude larger than any value not on the diagonal. The remaining values still contain important information to the classification task but the magnitude of the diagonal values make it difficult to any learning algorithm to make use of this information.

Weston et al. [55] discuss this typical problem and propose the *subpolynomial kernel*, which for an original kernel $K(x, y) = \langle \Phi(x), \Phi(y) \rangle$ where $\Phi(x)$ is a mapping into a high dimensional space uses the new kernel

$$K(x, y) = \langle \Phi(x), \Phi(y) \rangle^p$$

with $0 < p < 1$. Applied to a given kernel matrix this is an element-wise exponentiation with the exponent $p$.

The large diagonal is reduced, but the new kernel may not need to be positive definite. Weston et al. suggest to use the *empirical kernel map* from Tsuda [51] to properly enforce the positive definiteness of the new kernel matrix[12]:

$$K_m = KK^\top.$$

$K_m$ is the new positive definite kernel matrix, and $K$ is the original kernel matrix processed with the subpolynomial kernel. By construction, whatever $K$ is used, the resulting $K_m$ is guaranteed positive definite.

## 2.4 Marginalized Graph Kernel

A central objective of this thesis is to develop an effective way to represent natural objects as graphs. To use these graphs in a machine learning algorithm based on kernels, we employ a kernel on graphs, the *Marginalized Graph Kernel*. This section gives the theoretical background and the implementation of the kernel.

### 2.4.1 Graphs

Let a *directed graph* $G = (V, E)$ be a set of vertices (nodes) $V$ and a set of edges $E$ connecting those vertices. Let $|G| = |V|$ denote the number of vertices and $v_i$, $i = 1, \ldots, |G|$ be the individual vertices themselves, where $V := \{v_i\}_{i=1,\ldots,|G|}$. Similarly, let $e_{i,j}$, $i, j \in [1; |G|]$ be the *directed edge* from vertex $v_i$ to vertex $v_j$. Additionally every vertex and edge may have some further attributes of any kind be associated with them. The attributes together constitute the *label*, $L(v_i)$ and $L(e_{i,j})$ respectively. The resulting graph is called *attributed directed graph* or *labeled directed graph*.

An example with natural numbers as vertex attribute and a string as edge attribute is shown in figure 2.13.

[12] In practice, it may still be possible to train a SVM by using only the subpolynomial kernel.

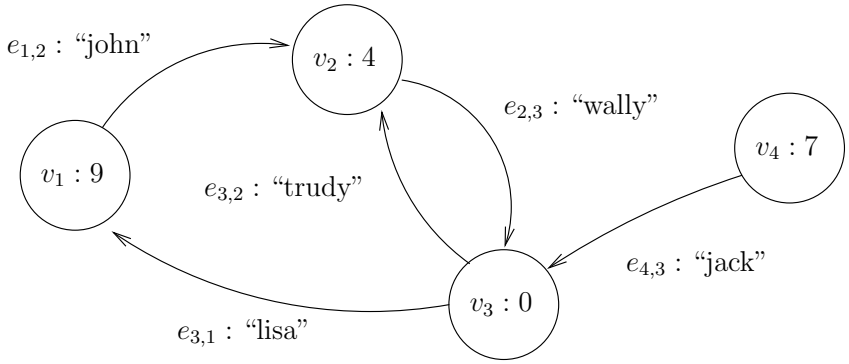


Figure 2.13: A directed attributed graph $G = (V, E)$, where $V = \{v_1, v_2, v_3, v_4\}$ is the set of vertices and $E = \{e_{1,2}, e_{2,3}, e_{3,1}, e_{3,2}, e_{4,3}\}$ is the set of edges.

### 2.4.2 Paths

Given a start vertex $h_1$, a *path* is a consecutive list of edges $g_i$ and vertices $h_i$ such that within the path, any edge $g_i$ connects the previous vertex $h_{i-1}$ with the successor vertex $h_i$. One example of a path in the graph shown in figure 2.13 given $h_1 := v_1$ is

$$(v_1, e_{1,2}, v_2, e_{2,3}, v_3, e_{3,2}, v_2, e_{2,3}, v_3).$$

We make the following two observations for the given example.

1. When starting with $h_1 = v_1$, the next edge must be $e_{1,2}$, because there is no other edge starting at $v_1$. Similarly, if the path continues, then $e_{2,3}$ must follow $v_2$. In $v_3$ there are two possible choices to continue the path, either to go along edge $e_{3,2}$ or along $e_{3,1}$.
2. With $h_1 = v_1$, we can never reach $v_4$ in the path.

Let $W$ be the set of all possible paths. Clearly, if $G$ contains any cycles, then $|W|$ is infinity. Let $L(p)$ be the *label path* of a path $p$ resulting of sequentially taking the labels of the path's vertices and edges, element-wise. For the above example, the label path would be

$$(9, \text{"john"}, 4, \text{"wally"}, 0, \text{"trudy"}, 4, \text{"wally"}, 0).$$

If we sample a large number of label paths $L(p)$ from two graphs, we could compare the set of samples. Intuitively, similar graphs produce similar label pathes.

By modeling every possible label path as *random variable*, a kernel can be defined as inner product of the count vector counting occurring label paths, averaged over all possible label paths. We now describe this kernel in detail, following Kashima et al. [25].

### 2.4.3 Marginalized Kernels

Given the hidden variables $\mathbf{h}, \mathbf{h}'$ and the visible variables $\mathbf{x}, \mathbf{x}'$, let $z = [\mathbf{x}, \mathbf{h}]$ and let $K_z(z, z')$ be the *joint kernel* depending on hidden and visible variables. If we know $p(\mathbf{h}|\mathbf{x})$, we can interpret it as a feature extractor on $\mathbf{x}$. Tsuda et al. defined the *marginalized kernel* [52] between $\mathbf{x}$ and $\mathbf{x}'$ as the expectation of the joint kernel over all possible values of $\mathbf{h}$ and $\mathbf{h}'$:

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} K_z(z, z') p(\mathbf{h}|\mathbf{x}) p(\mathbf{h}'|\mathbf{x}'). \tag{24}$$

### 2.4.4 Marginalized Graph Kernel

We now construct a kernel between two graphs. For this, let the hidden variable $\mathbf{h} = (h_1, \ldots, h_\ell)$ be a sequence of numbers $h_i \in [1; |G|]$ denoting vertex indices, where $v_{h_i}$ is the $i$'th vertex traversed in $\mathbf{h}$. To generate $\mathbf{h}$, we proceed as follows [25]:

1. $h_1$, the *starting* point of the random walk, is sampled from a distribution $p_s(h)$,
2. $h_i$, $i \geq 2$ is sampled from a transition probability $p_t(h_i|h_{i-1})$ or the random walk terminates with a probability of $p_q(h_{i-1})$.

For $p_t$ and $p_q$ we have the constraint

$$\sum_{j=1}^{|G|} p_t(j|i) + p_q(i) = 1. \tag{25}$$

The probability for the path $\mathbf{h}$ of length $\ell$ to appear on a random walk is

$$p(\mathbf{h}|G) = p_s(h_1) \prod_{i=2}^{\ell} p_t(h_i|h_{i-1}) p_q(h_\ell). \tag{26}$$

To use equation (24), we must further define the *joint kernel* $K_z$, which Kashima et al. [25] defines as product between kernel functions $K_v$ defined between vertices and $K_e$ defined between edges. For $\mathbf{z} = (G, \mathbf{h})$, $\mathbf{z}' = (G', \mathbf{h}')$ we have[13]:

$$K_z(\mathbf{z}, \mathbf{z}') := \begin{cases} 0 & \ell \neq \ell' \\ K_v(v_{h_1}, v'_{h_1}) \prod_{i=2}^{\ell} K_e(e_{h_{i-1}h_i}, e'_{h'_{i-1}h'_i}) K_v(v_{h_i}, v'_{h'_i}) & \ell = \ell' \end{cases} \tag{27}$$

That is, paths of different length always yield a joint kernel value of zero, while paths of the same length $\ell = \ell'$ are compared as product of aligned kernel evaluations between vertices and between edges.

Using equations (24 to 26), Kashima et al. [25] derive a system of equations that need to be solved to obtain $K(G, G')$. The simplest version uses an iterative scheme as follows. First some definitions:

$$\begin{aligned} q(h_i, h'_j) &:= p_q(h_i) p'_q(h'_j) \\ s(h_1, h'_1) &:= p_s(h_1) p'_s(h'_1) K(v_{h_1}, v'_{h'_1}) \\ t(h_i, h'_i, h_{i-1}, h'_{i-1}) &:= p_t(h_i|h_{i-1}) p'_t(h'_i|h'_{i-1}) K_v(v_{h_i}, v'_{h'_i}) K_e(e_{h_{i-1}h_i}, e_{h'_{i-1}h'_i}) \\ r_1(h_1, h'_1) &:= q(h_1, h'_1) \\ R_L(h_1, h'_1) &:= r_1(h_1, h'_1) + \sum_{i,j} t(i, j, h_1, h'_1) R_{L-1}(i, j). \end{aligned} \tag{28}$$

The kernel $K(G, G')$ can be computed from $s$ and $R_\infty$ as

$$K(G, G') = \sum_{h_1, h'_1} s(h_1, h'_1) \cdot R_\infty(h_1, h'_1). \tag{29}$$

There are two ways to obtain $R_\infty$ which we discuss in detail now.

[13] We use the original notation of Kashima [25], but occasionally number the graphs for clarity. Here, $G = G_1$ and $G' = G_2$.

**Iterative method.** Kashima et al. [25] showed that starting with $R_1 := r_1$ the iteration of equation (28) will eventually converge to the unique solution $R_\infty$. In practice, 20 to 40 iteration steps are required to obtain an accuracy of $1 \cdot 10^{-15}$.

**Solving a system of simultaneous linear equations.** In case $R_L$ converges for $L \to \infty$, at one point the following equilibrium must be reached:

$$R_\infty(h_1, h_1') = r_1(h_1, h_1') + \sum_{i,j} t(i, j, h_1, h_1') R_\infty(i, j).$$

Rewriting the conditions as matrix equation it becomes

$$R_\infty = r_1 + T R_\infty,$$

where $R_\infty, r_1 \in \mathbb{R}^{|G_1||G_2| \times 1}$ are row vectors and $T \in \mathbb{R}^{|G_1||G_2| \times |G_1||G_2|}$ is a coefficient matrix. This is a linear equation and can be solved for $R_\infty$ when rewritten as

$$(I - T) R_\infty = r_1.$$

We now explain the structure of the coefficient matrix $T$. For this, we first define a simple helper matrix $C \in \mathbb{N}^{|G_1||G_2| \times 2}$ as follows:

$$C := \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ \vdots & \vdots \\ 1 & |G_2| \\ 2 & 1 \\ \vdots & \vdots \\ 2 & |G_2| \\ 3 & 1 \\ \vdots & \vdots \\ |G_1| & |G_2| \end{bmatrix}.$$

The $T$ matrix can then be defined element-wise as

$$\begin{aligned} T_{i,j} &= t(C_{j,1}, C_{j,2}, C_{i,1}, C_{i,2}) \\ &:= p_t(C_{j,1}|C_{i,1}) \cdot p_t'(C_{j,2}|C_{i,2}) \cdot K_v(v_{C_{j,1}}, v'_{C_{j,2}}) \cdot K_e(e_{C_{i,1}C_{j,1}}, e_{C_{i,2}C_{j,2}}). \end{aligned}$$

Using the constraint on the sum of transition probabilities (25) we now prove that any row sum of $T$ is strictly smaller than one and larger or equal to zero. Let $i$ be any row index of $T$, but fixed. So $C_{i,1}$ and $C_{i,2}$ are fixed. Then we have

$$
\begin{aligned}
\sum_{j=1}^{|G_1||G_2|} T_{i,j} &= \sum_{j=1}^{|G_1||G_2|} p_t(C_{j,1}|C_{i,1}) \cdot p'_t(C_{j,2}|C_{i,2}) \cdot K_v(v_{C_{j,1}}, v'_{C_{j,2}}) \cdot K_e(e_{C_{i,1}C_{j,1}}, e_{C_{i,2}C_{j,2}}) \\
&\leq \sum_{j=1}^{|G_1||G_2|} p_t(C_{j,1}|C_{i,1}) \cdot p'_t(C_{j,2}|C_{i,2}) \qquad (30) \\
&= \sum_{j_1}^{|G_1|} \sum_{j_2=1}^{|G_2|} p_t(j_1|C_{i,1}) \cdot p'_t(j_2|C_{i,2}) \\
&= \sum_{j_1=1}^{|G_1|} \left( p_t(j_1|C_{i,1}) \cdot \underbrace{\sum_{j_2=1}^{|G_2|} p'_t(j_2|C_{i,2})}_{<1} \right) \\
&< \sum_{j_1=1}^{|G_1|} p_t(j_1|C_{i,1}) \\
&< 1.
\end{aligned}
$$

So any row sum $w_i$ of row $i$ in $T$ has $0 \leq w_i < 1$. The positivity can be easily seen in step (30), as the probabilities strictly positive the sum must be positive as well.

What is left to prove is the row sum of the matrix $I - T$. Clearly, the row sum of any row in $I$ is 1. Hence, for any row sum $0 \leq w_i < 1$ of $T$, we have for the row sum $u_i$ of the $i'th$ row in $I - T$:

$$0 < u_i \leq 1,$$

hence $u_i$ is strictly positive. For all practical uses we also have $0 < u_i < 1$, so no row sum exceeds one.[14] Then the system that must be solved has a coefficient matrix that is a non-singular asymmetric M-matrix[15], which can be solved efficiently using specialized solvers. Albeit interesting, the detailed examination of this aspect is outside of this thesis.

### 2.4.5 Implementation and use

An efficient general implementation of the Marginalized Graph Kernel in Matlab can be found in section A.2.

For practical application of the MGK, what remains to be defined are the two sub-kernels, one for the vertices and one for the edges and the three probability functions $p_s$, $p_t$ and $p_q$. Both the kernels and the probabilities are excellent points to incorporate prior knowledge into the graph kernel, as we will do in section 4.2. Without prior knowledge a sensible choice for the probabilities is

[14] We think this can only be the case when identical graphs are compared and the graphs are fully connected.
[15] Thanks to Dr. Christian Mense for pointing this out to me in personal communication.

$$
\begin{aligned}
p_s(i) &:= \frac{1}{|G|} \\
p_t(h_i|h_{i-1}) &:= \frac{1 - p_q(h_{i-1})}{|succ(h_{i-1})|} \\
p_q(h_i) &:= c,
\end{aligned}
$$

where $succ(h_{i-1})$ is the operation giving a set of all nodes that are successors of $h_{i-1}$. So $p_t$ produces an uniform probability and all outgoing edges are equally likely to be traversed. $p_s$ is uniform, so all nodes are considered uniformly to start the random graph walk. $p_q$ is set to a constant $c$ with $0 < c \leq 1$. Common choices are 0.1, 0.2, 0.5, but the choice depends very much on the problem at hand.

Kernels on graphs, their theoretical properties and practical applications are an active field of research. Straightforward extensions to the original Marginalized Graph Kernel can be found in Mahé et al. [35]. A theoretical analysis of graph kernels and computational complexity is Ramon and Gärtner [42].

# Chapter 3

# The set of keypoints model

In section 2.2.4 we have described a feature extraction mechanism for 2D natural images. Now we describe a simple yet popular model – the *set-of-keypoints* model – to work with such features in a machine learning context.

For this, we first introduce the model formally and then summarize previous work for object classification that work with this model. Finally, we summarize the problems resulting from the use of the model, which provides the basis for the next chapter.

## 3.1 Introduction

Let $F$ be a feature extraction function taking an image as argument and returning a list of individual features $f_1, \ldots, f_\ell$. Each feature $f_i$ consists of a feature data vector $d_i$ and feature meta data $m_i$.

1. Feature data vector $d_i$.

   For every individual feature detected in the image, information about the appearance of this feature is extracted and converted into a fixed length bounded vector $d_i \in \mathbb{R}^n$, where $n$ is the dimensionality of the feature vectors[1] and $\mathbb{R}^n$ is the *feature space*. Similar features are mapped to vectors close to each other in this feature space.

2. Feature meta data $m_i$.

   Data about the feature is aggregated in $m_i$. This commonly includes the position and shape of $f_i$ in the original image, as well as a scale parameter describing the size of the feature relative to the image size.

**Set-of-keypoints model.** The set-of-keypoints model discards all the meta information $m_i$ from the features. The retained data vectors $d_i$ are organized into a set

$$S = \{d_i\}_{i=1,\ldots,\ell},$$

the *set of keypoints*.

The idea behind the model can be explained by comparing it to the bag-of-words model popular in the information retrieval community. There, the original features extracted from

[1] Commonly $40 \leq n \leq 128$.

a natural language text are words and their positions in the text. The bag-of-words model discards all position information from the features, making the remaining information *invariant* to reordering of words in the original text: the bag-of-words has in fact become a word count histogram. Comparing such histograms can provide enough information for reliable text classification [24].

In our set-of-keypoints model things are similar but a little bit more complicated. Consider an average feature $f_i$, such as the one extracted by the SIFT method. There, $f_i$ has a $[0; 255]^{128}$ data vector, so a set-of-keypoints has no explicit bins anymore, but $2^{8 \cdot 128}$ possible elements, whereas a bag-of-words histogram only has one element for every possible word, a count not exceeding $100,000$ even for large text corpi.[2] Considering the original data vectors $d_i$ as simple histogram bins is infeasible because of the high dimensionality of $d_i$. One approach to counter this is discussed below, where a *bag-of-keypoints* is proposed that first reduces the set of features to a histogram count.

In the next section we will see that we can make use of the property that similar features are mapped to nearby feature data vectors to build similarity measures between two set-of-keypoints.

## 3.2 Literature: Set kernels for image classification

We now review literature for kernel functions between set-of-keypoints.

### 3.2.1 Wallraven, Caputo and Graf: Recognition with Local Features: the Kernel Recipe [54]

Wallraven, Caputo and Graf propose a kernel function

$$K(\mathbf{L}, \mathbf{L}') = \frac{1}{2}\left[\hat{K}(\mathbf{L}, \mathbf{L}') + \hat{K}(\mathbf{L}', \mathbf{L})\right] \tag{1}$$

with

$$\hat{K}(\mathbf{L}, \mathbf{L}') = \frac{1}{n}\sum_{j=1}^{n} \max_{i=1,\ldots,n'} \left\{K_l(\mathbf{l}_j, \mathbf{l}'_i)\right\},$$

where $\mathbf{l}_i \in \mathbf{L}$, $\mathbf{l}'_i \in \mathbf{L}'$ are the local features, $K_l$ is a kernel defined between two such features, and $n := |\mathbf{L}|$, $n' := |\mathbf{L}'|$. Unfortunately it was later discovered by Lyu [34] that $K$ is not a Mercer kernel because of the max-operator. Still, $K$ does perform reasonable well in practice.

Wallraven et al. test the kernel on three object recognition tasks. Unfortunately they do not describe the validation method to obtain their classification error value. Also, every object being tested was in the training set, but from a slightly different view, making it a pure recognition problem.

The results show that sets of local image features, compared with a suitable kernel function can outperform other approaches.

[2] Additionally words are often preprocessing by *stemming*, reducing words to their stems: "definitive" → "define", "definition" → "define", etc.

### 3.2.2 Boughorbel, Tarel and Boujemaa: The Intermediate Matching Kernel for Image Local Features [3]

Boughorbel et al. also pick up the original idea of the matching kernel. To fix the non positive-definiteness of the original match kernel [54], they propose to add a set of *virtual local features* $V$ which is used to determine two closest neighbors in each set to be matched. $V$ is fixed and chosen a-priori, for example by clustering all input set elements, but for discussing of the kernel that follows, consider it given.

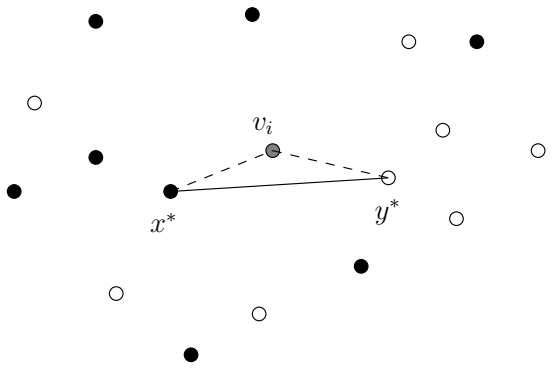


Figure 3.1: Intermediate Matching Kernel: for every virtual feature $v_i$ the two closest neighbors $x^*$ and $y^*$ from each set are determined, and the base kernel is evaluated on the distance $\|x^* - y^*\|$.

1. Produce mapped sets.

   Independently, for each set $X$ to be compared, do the following: for every virtual feature $v_i$ in $V$, determine the nearest element $x_i^* \in X$ such that $\|x_i^* - v_i\|$ is minimal among all $x \in X$. This is shown in figure 3.1. Then, let the set of all $x_i^*$, $y_i^*$ be called $X^*$ and $Y^*$ respectively.

2. Comparison of $X^*$ with $Y^*$.

   As both $X$ and $Y$ have been mapped to sets $X^*$ and $Y^*$ of equal cardinality $|V|$, the sets can be compared as simple as

   $$K(X,Y) = \sum_{i=1}^{|V|} \hat{K}(x_i^*, y_i^*),$$

   where $\hat{K}(x_i^*, y_i^*)$ is any valid kernel defined on the elements of $X$ and $Y$, such as the Gaussian RBF.

This kernel is provably positive definite. This is easy to see if you consider that $V$ is chosen a-priori and every set is preprocessing with it independently of any other set.

Experimentally, they test their kernel on a recognition task where a large number of samples are available and achieve moderate results, showing no improvement over the original match kernel.

While the idea of their kernel looks interesting on the surface, the crux of the whole approach lies in the choice of the virtual feature set $V$. The features in $V$ have to be chosen a-priori, where the best choice are features in highly informative and dense areas in feature space. In case enough samples are available this can be done using clustering, as Boughorbel et al. propose. But choosing both the number of virtual features and the virtual features themselves on only the training samples could lead to uninformative virtual features for the testing samples.[3]

Additionally, the clustering Boughorbel et al. propose can be difficult in high dimensional spaces such as the one SIFT features live in. Boughborbel et al. test their kernel on nine dimensional Jets where both nearest neighbor matching and clustering are efficient. In high dimensional spaces nearest neighbor matching is expensive and clustering is more difficult.

Summarizing, while the idea of choosing "prototype" features a-priori to construct a kernel or to incorporate prior knowledge is nice and could be useful to construct kernels in other problem settings, for local image features both the high dimensionality and variability of local image features within one object or object class make it inefficient.

A second earlier attempt to "fix" the Match kernel by the authors is Boughorbel et al. [4], where they propose a probabilistic Match kernel that with a high probability produces a positive definite kernel matrix; but again the recognition accuracy does not exceed the one of the match kernel.

### 3.2.3 Grauman and Darrell: Pyramid Match Kernel: Discriminative Classification with Sets of Image Features [20]

Grauman and Darrell propose a general kernel for unordered sets of vectors, where the sets can have different cardinality. We now describe their kernel in detail, as it is one of the most efficient kernels for sets of vectors proposed so far and computationally the fastest.

Let $\mathbf{L} = \{\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_n\}$ be the set of features $\mathbf{l}_i \in \mathbb{R}^d$, where $d \in \mathbb{N}$ is the dimensionality of the feature vectors and $n := |\mathbf{L}|$ is the cardinality of the set.

Let the feature vectors $\mathbf{l}_i$ have a minimum distance between each other of *one* and let $D$ be the diameter of a sphere on the origin containing all samples in $\mathbf{L}$. This can be achieved simply by properly scaling the input data uniformly.

Define a feature extraction function

$$\Psi(\mathbf{L}) = [H_{-1}(\mathbf{L}), H_0(\mathbf{L}), H_1(\mathbf{L}), \ldots, H_k(\mathbf{L})]\,,$$

where $k := \lceil \log_2(D) \rceil$ is the pyramid level count minus two and $H_i(\mathbf{L})$ is a histogram vector over $d$-dimensional bins, each having a side length of $2^i$ for each dimension.

To understand the idea of this recursive histograms better, lets take a look at an example of samples in $\mathbb{R}^2$, shown in figures 3.2 to 3.6.

1. Figure 3.2 shows the original data in $\mathbb{R}^2$. The axes are uniformly scaled such that the minimum Euclidean distance between any two samples is one.[4] Here, $D = 4$.

2. Figure 3.3 shows the finest histogram $H_{-1}($**L** with all bins marked by the dashed lines. The length of the bin in each dimension is $\frac{1}{2}$ and so every sample is guaranteed its

[3] Using unlabeled testing samples in a transduction setting could improve this.

[4] For features like SIFT this is not even necessary, as SIFT vectors live in $\mathbb{N}^{128}$ and except for identical vectors any two vectors always have a distance equal to or larger than one.

own bin. The bins containing at least one sample are marked gray in this and the next figures. Note that the histogram is sparse, as the number of bins with samples can never exceed $n$, the number of samples. This makes a sparse, linear and strictly ordered representation of the histogram possible even when $d$ is very large.

3. The histogram $H_0(\mathbf{L})$ is shown in figure 3.4. The bin size has doubled in each dimension, hence the bin side length is one. Most samples are still likely to be the only samples occupying their bin, but it is possible for two or more samples to be in the same bin, as shown in the bottom left of figure 3.4.

4. $H_1(\mathbf{L})$ once again doubles the bin size in each dimension, shown in figure 3.5. Now, for our example data all bins are occupied, albeit with a different number of samples.

5. $H_2(\mathbf{L})$ in figure 3.6 is the coarsest trivial one-bin histogram with a bin length of $2^i = 2^2 = D$. All samples are mapped to the same bin.

The pyramid match base kernel is a function between two such multi-resolution histograms $\Psi(\mathbf{L})$ and $\Psi(\mathbf{L}')$ and is defined as weighted sum of the bin-wise feature matching count in each resolution:

$$\tilde{K}_\Delta(\Psi(\mathbf{L}), \Psi(\mathbf{L}')) := \sum_{i=0}^{k} w_i N_i.$$

$N_i$ is the number of *newly* matched features, defined as

$$N_i := I(H_i(\mathbf{L}), H_i(\mathbf{L}')) - I(H_{i-1}(\mathbf{L}), H_{i-1}(\mathbf{L}')),$$

where $I$ is an intersection function between two histograms $\mathbf{A}$, $\mathbf{B}$ with $c$ bins, defined as

$$I(\mathbf{A}, \mathbf{B}) := \sum_{i=1}^{c} \min(\mathbf{A}(i), \mathbf{B}(i))$$

measuring the *overlap* between the histograms $\mathbf{A}$ and $\mathbf{B}$. What remains to be defined is $w_i$, the weightings. Clearly, a match on a fine scale should count more than a match on a coarse scale with large-sized bins. Hence $w_i$ is defined as

$$w_i := \frac{1}{2^i}.$$

To avoid favoring sets with larger cardinality the standard normalization from section 2.3.4 is applied to obtain the final real *Pyramid Match Kernel* $K_\Delta$ as

$$K_\Delta(\Psi(\mathbf{L}), \Psi(\mathbf{L}')) = \frac{\tilde{K}_\Delta(\Psi(\mathbf{L}), \Psi(\mathbf{L}'))}{\sqrt{\tilde{K}_\Delta(\Psi(\mathbf{L}), \Psi(\mathbf{L}))\tilde{K}_\Delta(\Psi(\mathbf{L}'), \Psi(\mathbf{L}'))}}. \tag{2}$$

Grauman reports results on a proper classification task on the ETH80 "Eichhorn-A" subset used in Eichhorn and Chapelle [13]. The classifier is trained in a leave-one-object-out fashion and the final error is obtained via cross validation. The best result on the relatively difficult set with an average of 153 features per image was obtained as 82%, which is quite impressive. Additionally this kernel is two orders of magnitude faster than any other kernel for sets of vectors. Even more, additional vectors in one set do not lead to substantially lower kernel values, leading to clutter resistance.

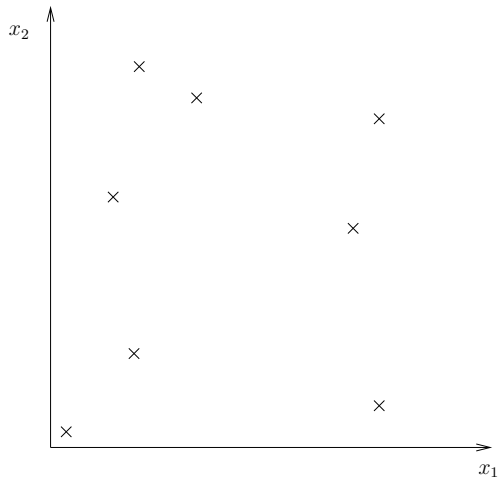


Figure 3.2: The original data points in $\mathbb{R}^2$.

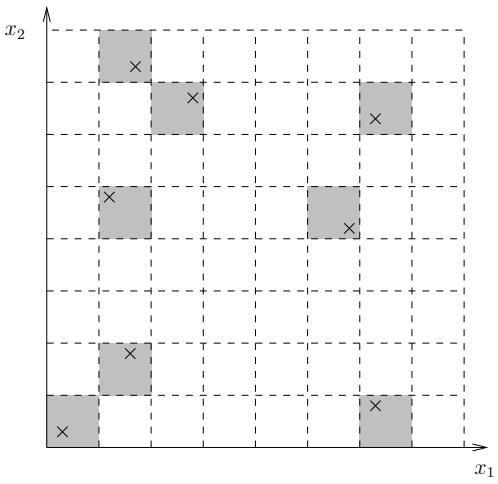


Figure 3.3: Pyramid histogram $H_{-1}$: bin length equal to half the minimum inter sample distance.

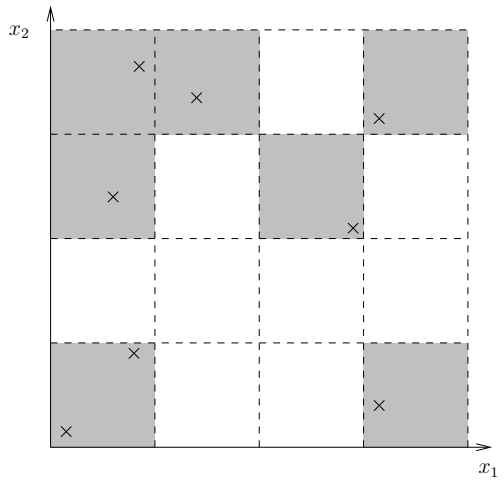


Figure 3.4: Pyramid histogram $H_0$: bin length equal to minimum inter sample distance.

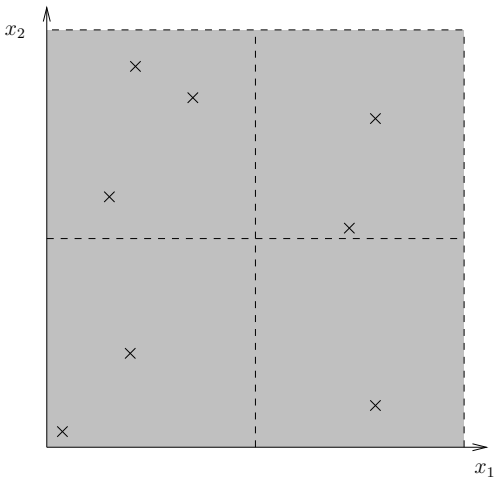


Figure 3.5: Pyramid histogram $H_1$: only four remaining bins.

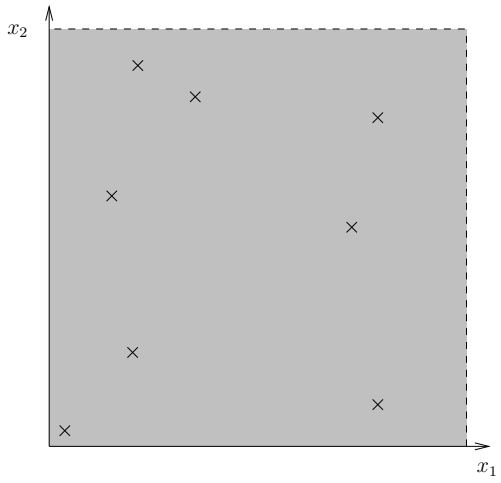


Figure 3.6: Pyramid histogram $H_2$: the trivial histogram with only one remaining bin.

Summarizing, the Pyramid Match Kernel is one of the most promising kernels for general vector-of-set problems. However, for object classification by local image features it is still subject to all the limitations resulting from the set-of-keypoints model. These problems will be examined in the next section.

It may be interesting to examine the relationship between the Pyramid Match Kernel and the Earth Mover's Distance (EMD) [10] or the Proportional Transportation Distance (PTD) [17], as the Pyramid Match Kernel's use of histograms resembles a discretized distance measure between the features, whereas both the EMD and PTD use Euclidean distances as a ground distance to establish optimal assignments between features of each set. Perhaps it is possible to define a sensible kernel upon the non-metric EMD.

### 3.2.4 Lyu: Mercer Kernels for Object Recognition with Local Features [34]

Lyu proposes a kernel based on the matching kernel by Wallraven et al. [54]. After its publication, the original match kernel in equation (1) was discovered to be a non-Mercer kernel due to the max-Operator. Lyu proposes to replace $\hat{K}(\mathbf{L}, \mathbf{L}')$ with

$$\hat{K}(\mathbf{L}, \mathbf{L}') = \frac{1}{n} \sum_{j=1}^{n} \left[ \frac{1}{n'} \sum_{i=1}^{n'} \left( K_l(\mathbf{l}_j, \mathbf{l}'_i) \right)^p \right], \tag{3}$$

so the max-Operator in equation (1) is effectively replaced by the inner term. This works due to

$$\max_{i=1,\ldots,n'} \left\{ K_l(\mathbf{l}_j, \mathbf{l}'_i) \right\} \approx \sqrt[p]{\frac{1}{n'} \sum_{i=1}^{n'} \left( K_l(\mathbf{l}_j, \mathbf{l}'_i) \right)^p}$$

in case one kernel value $K_l(\mathbf{l}_j, \mathbf{l}'_i)$ is sufficiently larger than all other values, which can be expected in case there is a good match. In equation (3) the root is not applied as only the relative contribution of a good match is important. Lyu recommends a high power $p \geq 9$ as then a single good match contributes more than 99% of the total kernel value, resembling the max-Operator closely. The exponentiation instead of max makes the kernel a Mercer kernel.

Lyu is the first to break out of the strict set-of-keypoints model when designing his kernel: he incorporates more than one type of local feature and adds local geometry constraints on the image feature positions.[5]

**Multiple feature types.** Let every local feature $\mathbf{l}_i$ be a tuple of local features $\mathbf{l} := (\mathbf{l}_i^{(1)}, \mathbf{l}_i^{(2)}, \ldots, \mathbf{l}_i^{(f)})$, then $K_l$ can be defined as product kernel from section 2.3.4, such that

$$K_l(\mathbf{l}_i, \mathbf{l}'_j) := \prod_{m=1}^{f} K_l^{(m)} \left( \mathbf{l}_i^{(m)}, \mathbf{l}_j^{\prime(m)} \right).$$

[5] Note that such geometric constraints are very popular in object recognition, but Lyu is the first to incorporate them into a kernel function while still preserving all desired invariances. For example, Wallraven et al. [54] also propose to incorporate positions of keypoints into the kernel, however – as Lyu points out – they use absolute positions corrupting most invariances.

**Local geometry constraints.** The simple set-of-keypoints model is invariant against changes of the positions of the feature points in the image. But the *relative positions* of the features contain valuable information for most classification and recognition tasks. For example, the tires of a car are always positioned in a fixed pattern relative to each other – in rectangular or triangle corners – and to the rest of the car, almost always at the bottom side of the car.

Lyu incorporates the semi-local constraints from Schmid and Mohr [44] into his kernel function $K_l$. He replaces $K_l(\mathbf{l}_i, \mathbf{l}'_j)$ with a kernel defined on semi-local groups $\mathbf{g}_i = \{\mathbf{L}^{(i)}, \Theta_i\}$. $\mathbf{L}^{(i)} \subseteq \mathbf{L}$ is an ordered set of local features around a center feature point and $\Theta_i = (\theta_1, \ldots, \theta_{|\mathbf{L}^{(i)}|})$ is a tuple of angles between such neighbor points. This is shown schematically in figure 3.7.

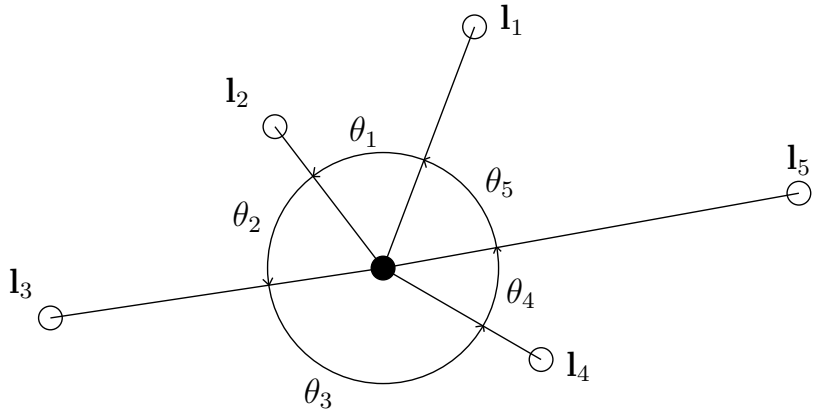


Figure 3.7: Semi-local constraints by using local groups: around a central feature a set of geometrically closest neighbor features $\mathbf{l}_i$ are selected. The angles $\theta_i$ between two such features relative to the center point produces the tuple $\Theta$.

Instead of comparing sets of local features, the original kernel is modified to compare *sets of groups* of local features $\mathbf{G} = \{\mathbf{g}_1, \ldots, \mathbf{g}_{|\mathbf{L}|}\}$. The kernel (3) becomes

$$\hat{K}_G(\mathbf{G}, \mathbf{G}') = \frac{1}{n} \sum_{j=1}^{n} \left[ \frac{1}{n'} \sum_{i=1}^{n'} \left( K_g(\mathbf{g}_j, \mathbf{g}'_i) \right)^p \right].$$

Additionally, $K_g$ needs to be specified. While the relative order of the angles in $\Theta$ is fixed and the angles shall be compared in this order when matching two local groups, the angle to start comparing with – the alignment of two $\Theta$-vectors – is unclear. To give an example, consider the case where the indices in figure 3.7 are rotated by one, that is the top point is now named $\mathbf{l}_5$ and the point left of it is $\mathbf{l}_1$ and so on. Clearly, the new group is still representing the same constellation of features in the image as the old one and should match well with it.

To achieve this rotation invariance, Lyu defines a circular shift operator $(c(y, q))_i := (y)_{(q+i) \bmod |y|}$. Then, $K_g$ can be defined as

$$K_g(\mathbf{g}, \mathbf{g}') = \hat{K}(\mathbf{L}, \mathbf{L}') \cdot \sum_{q=0}^{|\mathbf{L}|-1} \left[ K(\Theta, c(\Theta, q)) \right]^p,$$

where $K$ is any kernel defined on vectors, such as the Gaussian RBF kernel from section 2.3.4. $\hat{K}(\mathbf{L}, \mathbf{L}')$ is the match kernel defined in equation (3).

Lyu tests his kernel on three kinds of local image features in an object recognition setting, reporting a good recognition accuracy on the COIL-100 database.

### 3.2.5 Kondor and Jebara: A Kernel Between Sets of Vectors [27]

In [27] Kondor and Jebara propose a kernel between sets that is actually a kernel between Gaussian distributions fitted to these sets. Because they do not apply it to local image features in our sense and are more concerned with the general setting of comparing sets, we only briefly summarize their interesting work. A similar approach which we do not discuss here was independently discovered by Wolf and Shashua [58].
Their approach is based on the following ideas.

1. The vectors in a set of vectors can be mapped to a high dimensional Hilbert space $\mathcal{H}$ using a kernel function such as the Gaussian RBF kernel from section 2.3.4.

2. Simple parametric distributions in $\mathcal{H}$ can capture complex structures in the original input space of the vectors.

3. The Bhattacharyya affinity[6] between probability distributions in $\mathcal{H}$,

$$K(p, p') = \int_{\mathcal{H}} \sqrt{p(z)}\sqrt{p'(z)}dz$$

   can be i) shown to be a valid kernel function, and ii) computed in closed form without explicitly mapping the input vectors into $\mathcal{H}$.

4. To avoid overfitting the distributions in the possibly infinite-dimensional space $\mathcal{H}$ Kernel PCA [45] is used to keep only the first few important eigenvectors of the distribution's covariance matrix, effectively regularizing the resulting distributions.

Kondor and Jebara further make an interesting point to consider when constructing kernels: because their method compares sets and those sets can consist of tuple spaces, say $(x, y, i)$ for an image pixel at $(x, y)$ with an intensity of $i$, it can behave smoothly in all these three coordinates. A standard representation, say only the intensity values of an image concatenated in a large vector, cannot behave smoothly in the ordering of the vector's elements, which includes desirable invariances against scaling, translation and rotation.

### 3.2.6 Moreno, Ho and Vasconcelos: A Kullback-Leibler Divergence Based Kernel for SVM Classification in Multimedia Applications [40]

Moreno et al. propose a kernel based on the Kullback-Leibler divergence between probability density functions (PDF). The probability density functions are Gaussian Mixture Models (GMM) estimated from a set of vectors.[7]

[6] A. Bhattacharyya, 1943. "On a measure of divergence between two statistical populations defined by their probability distributions.", Bulletin Calcutta Mathematical Society, 35, 99-110. Despite hundreds of references I found the original paper to be impossible to locate.

[7] Throughout their paper Moreno et al. ambiguously refer to "sequence of vectors" but nowhere the order of vectors is incorporated into the kernel. Only in their speaker recognition experiment setup they describe that they annotate vectors by first and second order time derivatives using the order property, but this has nothing to do with the workings of the kernel.

The approach can be divided into two steps, the estimation of the probability density function $p(\mathbf{x}|\theta_i)$ for each sample and the kernel evaluation between two such functions.

**Estimation of $p(\mathbf{x}|\theta_i)$.** To compute the PDF parameters $\theta_i$ for a set $\mathbf{L}$, Moreno et al. use a maximum likelihood approach, either – for simple mixture models – by analytical solution or – for diagonal mixture models – numerically by using the Expectation Maximization algorithm. This effectively maps $\mathbf{L}$ to a new feature space $\theta_i$, so every original sample $\mathbf{L}$ is represented by an equal sized vector.

**Computing the kernel.** The *symmetric Kullback-Leibler divergence* is used to evaluate a distance $D$ between two PDFs as

$$D(p(\mathbf{x}|\theta_i), p(\mathbf{x}|\theta_j)) = \int_{-\infty}^{\infty} p(\mathbf{x}|\theta_i) \log\left(\frac{p(\mathbf{x}|\theta_i)}{p(\mathbf{x}|\theta_j)}\right) dx + \int_{-\infty}^{\infty} p(\mathbf{x}|\theta_j) \log\left(\frac{p(\mathbf{x}|\theta_j)}{p(\mathbf{x}|\theta_i)}\right) dx. \quad (4)$$

This distance function is not a strict metric and hence a kernel matrix based on such distances is not automatically positive-definite, as shown by Haasdonk and Bahlmann [21]. Moreno et al. suggest to use a Gaussian RBF distance substitution kernel on the scaled and shifted distances $D(p(\mathbf{x}|\theta_i), p(\mathbf{x}|\theta_j))$, but this is an ad-hoc empirical fix and does not guarantee Mercer's condition.

While for the single full covariance models there is a closed form for the solution of $D$, for the GMM estimated in the first step the computation of (4) must be carried out numerically by Monte Carlo methods or other methods.

One of the two experiments reported by Moreno et al. is an image classification task with features based on shifting a window over the image and extracting for each color channel the first 64 low frequency elements extracted by the Discrete Cosine Transform. On a self selected subset of the COREL database with eight classes, they report an accuracy of 85.3 percent.

However, the type of feature used to extract information from the images is highly likely to let the SVM actually learn color distributions of each class. This leads to good classification accuracy but may not be able to correctly classify the object in a new, possibly cluttered setting.

### 3.2.7 Csurka, Dance, Fan, Willamowski, Bray: Visual Categorization with Bags of Keypoints [11]

Csurka et al. [11] propose to reduce each set of keypoints extracted for each image to a fixed size cluster histogram count vector, the *bag-of-keypoints*. The set of clusters is determined a-priori from the overall set of features of all training images by $k$-means clustering, where the number of clusters is set to large value $k = 1000$.

After the histogram vectors have been created, each image is represented only by the fixed size histogram vector and a standard multiclass SVM can be used to learn the training set.

The results reported are impressive for a large self-compiled training set in a pure object classification setting. One possible problem in the experiments reported is that its unclear whether the entire training set was once used to determine the clusters prior to evaluating a cross-validated error rate, or if proper care was taken to only use the training data of each fold to build the clusters.

### 3.2.8 Farquhar, Szedmak, Meng, Shawe-Taylor: Improving "bag-of-keypoints" image categorisation: Generative Models and PDF-Kernels [14]

The approach of Farquhar et al. is a generalization on Csurka et al. [11]. In the original approach a cluster histogram is build, counting the cluster membership of the features in a histogram. If the features are assumed to have been produced by a probability density function (PDF) in a Gaussian mixture model (GMM), where the original clusters correspond to one component in the mixture model, then the histogram would only be an approximation to the partial generative component responsibilities of each feature. Thus, Farquhar et al. [14] propose to replace the histogramming with fitting of a GMM using Expectation Maximization (EM). To counter overfitting, the feature dimension is reduced using Principal Components Analysis (PCA) or Partial Least Squares (PLS) on a per-class basis. This already improves the classification on the same dataset as in [11], but additionally using direct PDF kernels, the Kullback-Leiber divergence from [40] and Bhattacharyya's affinity from [27] leads to an even larger improvement. However, because the PDF kernels for the used GMM have an analytical solution only for the case if one component is used, the potential of the mixture model is not fully exploitable.

## 3.3 Literature: comparison

With all the discussed approaches on the table, which one is the best? There is no thorough comparison of all the approaches above, but one early comparison study of three approaches is available, which we discuss now.

### 3.3.1 Eichhorn and Chapelle: Object categorization with SVM: Kernels for Local Features [13]

The first thorough evaluation of competing set-of-keypoints object classification approaches has been carried out by Jan Eichhorn and Olivier Chapelle. They compare the original Matching kernel [54], the Bhattacharyya PDF kernel [27] and the Kernel Principal Angles approach [58] on Scale-Invariant Feature Transform, JET and 6x6 pixel image patch features.

Further, Eichhorn and Chapelle define two subsets of the ETH80 object classification benchmark set[8], one with high variability in pose (set A) and one with a low variability (set B). We discuss these subsets in detail in section 5.1.

For the tested approaches, Eichhorn and Chapelle report the best results for an average of 40 detected features per image for SIFTs and JETs as shown in table 3.1.

## 3.4 Summary

We have discussed various approaches to object classification using set kernels. The choice of the set kernel to use in a classification system depends on their different qualities. Following Grauman and Darrell [20], we use the following qualitative attributes to contrast the approaches.

**Runtime complexity.** The runtime complexity of the discussed algorithms is given in Big-O notation, depending on the set cardinalities and algorithm-specific parameters. Note

[8] `http://www.mis.informatik.tu-darmstadt.de/Research/Projects/categorization/eth80-db.html`

Table 3.1: Classification success rate results on ETH-80 subsets as reported by Eichhorn and Chapelle [13]: both the matching and Bhattacharyya kernel provide good results with SIFT features, the Bhattacharyya kernel outperforms all other kernels for JETs and image patches. The KPA approach consistently produces bad results.

| Method | SIFT | JET | 6x6 patches |
|---|---|---|---|
| Matching [54] | 72% | 43% | 46% |
| Bhattacharyya [27] | 74% | 70% | 74% |
| Kernel Principal Angles [58] | 27% | 24% | 23% |

that the complexity is only important when considering scaling the dependent parameters. The actual implementation can be faster or slower.

**Co-occurrences.** The ability of the kernel function to recognize the additional value of co-occurring vectors within the set. Beyond existence of this property, the extend to which the kernel values co-occurence is difficult to quantify without detailed examination.

**Positive-definiteness.** As kernels are not trivial to design some kernels were later to be discovered to not fulfill Mercer's condition from section 2.3.4, despite being successful in experiments. It is risky to use such kernels or to build upon them as many theoretical guarantees for SVM vanish, such as convergence to or existence of a unique solution.

**Unequal set cardinalities.** A set kernel should be able to compare sets of different cardinalities, which is one of the purposes we need a set kernel in the first place.

**Geometry constraints.** A general set-of-vector kernel is useful for many applications, but for object classification tasks such kernel can improve if they incorporate known geometry relationships between the elements of the sets. This is not possible in the standard set-of-keypoints model and presently only the kernel of Lyu [34] extends the model.

Table 3.2 is an extension from [20] evaluating the properties of all the discussed approaches.

## 3.5 Problems of the set-of-keypoints model

The set-of-keypoints model only captures the feature data vectors and their co-occurrence information. Beside this loss of information, it additionally has the following shortcomings.

1. Uniform weighting of features.

   All features in the set are considered equally important. While a machine learning algorithm could learn such weights from the training data, the model itself provides no helpful information about the feature's relative importance in the image.[9] The bag-of-keypoints approaches which use a prior clustering step implicitly change the weights of the features, but this can only be seen as an approximation based on the assumption that higher density regions in feature space correspond to relevant information for the classification problem, which does not have to be the case.

[9] Of course it would be difficult to determine "importance" for the task at hand but clearly relative size of a feature and saliency within the image are important for most tasks.

Table 3.2: Set of keypoints kernels summary table. $m$ is the maximum cardinality of the two sets to be compared. $D$ is the diameter of the feature space. Remarks: i) Match kernel: the geometry constraints used by Wallraven et al. [54] are absolute positions, destroying desired invariances of local image features, ii) Intermediate match kernel: $p$ is the number of virtual features in $V$, the co-occurrence properties depend on the choice of $V$, iii) Bags-of-keypoints: $p$ is the number of clusters used, unequal cardinalities are not well handled because the histogram vector norm depends on the set cardinality, iv) GMM and PDFs, complexity would be $O(1)$ but this does not include MAP-EM fitting for each sample and only works for GMMs with one component, Kullback-Leibler divergence is not necessarily positive-definite, but the second tested Bhattacharyya kernel is.

| Method | Runtime complexity | Captures feature co-occurrences | Positive-definite | Model-free | Handles unequal cardinalities | Geometry constraints |
|---|---|---|---|---|---|---|
| Match kernel [54] | $O(dm^2)$ | | | √ | √ | (√) |
| Exponent match [34] | $O(dm^2)$ | | √ | √ | √ | √ |
| Greedy match [4] | $O(dm^2)$ | √ | | √ | √ | |
| Intermediate [3] | $O(pm)$ | (√) | √ | | √ | |
| Bags-of-keypoints [11] | $O(pm)$ | √ | √ | | (√) | |
| GMM and PDFs [14] | see comment | √ | (√) | | √ | |
| Principal angles [58] | $O(dm^3)$ | √ | √ | | | |
| Bhattacharyya [27] | $O(dm^3)$ | √ | √ | | √ | |
| KL divergence [40] | $O(dm^2)$ | √ | | | √ | |
| Pyramid match [20] | $O(dm \log(D))$ | √ | √ | √ | √ | |

2. Discarding shape information.

   By discarding the positions of the features in the original image, the overall shape information of an object is lost. This contrasts with the natural way we look at things: not only *what* we see is important, but *where* we see it in *what visual surrounding* is important as well.

   There is an exception to the above statement. As Sivic et al. [48] note, some information about spatial relationships remains as the feature regions are allowed to overlap. The mutual information in the overlapped region thus implicitly encodes the relative positions.

3. Discarding other meta information.

   Two other common meta attributes of a keypoint are produced by most modern scale-space based feature extractors, the *scale* and *orientation* of a keypoint. We cannot use these values absolutely for every individual keypoint, because they depend on the size and rotation of an image; both are properties we want to be invariant against.

   The set-of-keypoint model does not allow one to make use of this information because only individual keypoints remain and relationships between pairs of keypoints do not exist. We will see in the next chapter, that we can make good use of this information by using relative differences of scale and orientation between *pairs of keypoints*, while

retaining all invariances.

# Chapter 4

# A better model: graphs of local features

In this chapter we introduce our novel approach to the object classification problem. Cast as natural extension of set based methods, our model captures more information and is likely to show improved classification results. This claim is validated in chapter 5.

## 4.1 Introduction

Graphs can be seen as a straightforward extension to a set representation of local image features. But where the set representation isolates each feature, the graph allows us to encode valuable information *between* individual features. Relative position, size and orientation of the features are obviously all relevant to object classification tasks.

Hence, our *hypothesis* is that if we can incorporate this information into the graph then we can improve classification performance when using a graph kernel respecting this information. Given the freedom we have in defining the graph, the information could be added at two possible places, i) the graph structure itself could be determined depending on the inter-features information, or ii) the information can be added as attributes to edges.

The problem in the first approach is the difficulty to specify a graph construction method that works well for a wide variety of different object classes. Different object classes might have completely different characteristics; a descriptive graph structure depends on the object class. For example, a graph for "faces" might consist of local links between the roughly equal-sized facial features, such as eyes, nose, ears, mouth, etc. Contrasting, a graph for "plants" would have a much different structure: fine elements such as leafes have to be put in relation with coarse structures such as the trunk or overall shape of the plant. For other kinds of learning systems, it may be possible to derive per-class graph construction methods from the training data and then evaluate all available methods on a new testing image, chosing the best applicable one. In the kernel based approach we consider here, this is not possible.

In the second approach however, this problem can be circumvented by assuming a trivial fully connected graph structure where all edges are attributed with inter-feature information. We choose the second approach for our general object classification problem.

Evaluating the Marginalized Graph Kernel (24) on such attributed, fully connected graphs can then be seen as computing the closed form of a two stage classification system: the random walks within the graph generate hypotheses of all possible feature combinations within the

image, and the kernel functions $K_v$, $K_e$ act as "filters" which suppress those hypotheses that do not occur in both images. The MGK value is then the sum of all matching hypotheses.

The remainder of this section describes how such graphs are constructed. The Marginalized Graph Kernel is then used between the graphs to provide the Gram matrix of the SVM classifier.

## 4.2 The graph-of-keypoints model

Given the set of features extracted from an image, constructing a discrete graph from the set involves chosing suitable edges and attributes. The resulting graphs shall be robust to continuous changes in the image, such as rotations or intra-class variances. For the reasons outlined above, we use a simple uniform transition probability $p_t(v_j|v_i) = \frac{1-c_q}{|G|-1}$ for $i \neq j$ and $p_t(v_j|v_i) = 0$ for $i = j$. $c_q$ is the constant termination probability, explained below.

This feature graph structure we use is shown schematically in figure 4.1. The attributes and probabilities at the vertices and edges are described in the following paragraphs.

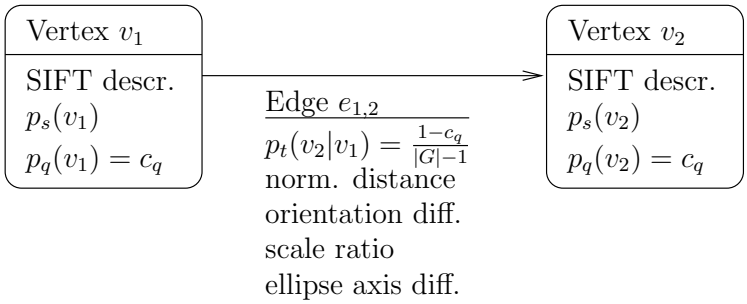


Figure 4.1: Graph information diagram. Each vertex $v_i$ has a SIFT descriptor, a start probability $p_s(v_i)$ and a constant termination probability $p_q(v_i) = c_q$ associated with it. Each edge $e_{i,j}$ has a constant transition probability $p_t(v_j|v_i) = \frac{1-c_q}{|G|-1}$ and four attributes derived from the two adjacent vertices.

### 4.2.1 Vertices

Each interest point is represented by one vertex. For performance reasons, we explicitly reduce the number of vertices as follows. The point with the smallest scale is removed iteratively from the set until the maximum allowed number of points is remains. From the remaining points the graph is build and the interest point descriptor is used as label for the vertices.

The probability $p_s(v_i)$ to start the random walk at the vertex $v_i$ is computed as

$$p_s(v_i) = \frac{\text{scale}(v_i)^{\alpha_s}}{\sum_j \text{scale}(v_j)^{\alpha_s}},$$

which biases the random walk to more likely start at vertices with a larger scale. If $\alpha_s = 0$ every vertex has equal probability, for $\alpha_s = 1$ the probability is linear to scale. The bias to prefer larger scale features is strongest when $\alpha_s$ is large.

The idea behind biasing the start probabilities for larger scales is to better exploit the addition of small scale detail features. If no such bias is used, that is if $\alpha_s = 0$, the addition of

these features reduces the probability of paths starting at large scale features. Our assumption is that large scale features are more likely to be shared across objects within the same class. If the start probability is chosen such that these paths are still likely to appear when small scale features are added, then we can increase the classification rate with additional features. Experimentally, a good value turned out to be $\alpha_s = 3$.

For the termination probability $p_q(v_i)$ we use a constant termination probability $p_q(v_i) = c_q$ for all $i$. Experimentally, $c_q = 0.5$ was determined as an appropriate value.

The vertex subkernel $K_v$ comparing the 128 dimensional SIFT descriptors is a Gaussian RBF kernel with a free parameter $\sigma$. For most experiments, $\sigma = 250$ provides good results.

### 4.2.2 Edges

Consider two features $v_i$, $v_j$ produced by the interest point detector from an image. The descriptor and detection method we use provides meta data for each image feature.

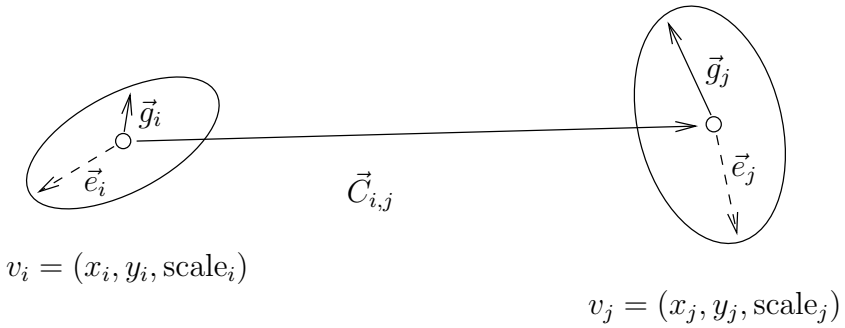


Figure 4.2: Available feature meta data at each interest point: the position in scale space $(x, y, \text{scale})$, the gradient orientation vector $\vec{g}$ and the major ellipse axis vector $\vec{e}$.

The meaning of the available feature meta data is shown schematically in figure 4.2. The following meta data is available for each feature.

- The position $(x, y)$ the feature was detected at within the image. This position is provided at sub-pixel accuracy by the standard method of fitting parametric functions around the neighborhood pixels.

- The characteristic $\text{scale}_i := \text{scale}(v_i)$ of the interest point $v_i$. This scale is the third coordinate to uniquely address the interest point in the scale-space over the image. Compare with section 2.2.2.

- A parametrized elliptical region around the interest point. The affine structure estimation of [37] provides the values of these parameters, effectively fitting an ellipse around the neighborhood of an interest point. The coefficients $a, b, c$ uniquely define the ellipse implicitly by the elliptic equation $a(u-x)^2 + 2b(u-x)(v-y) + c(v-y)^2 = 1$, where $(u, v)$ anchors a relative coordinate system at the interest point. Characteristic data of this ellipse can be used to coarsely describe the affine structure in the vicinity of the point. One such characteristic is the *major axis direction* $s_i$, which can be obtained through

$$s_i = \frac{1}{2}\tan^{-1}\left(\frac{2b}{c-a}\right). \tag{1}$$

In figure 4.2, the major axis direction is indirectly shown by the direction of the vector $\vec{e}$. It is also possible to calculate the eccentricity of the ellipse, but we have not done so.

- The major gradient orientation $\vec{g}_i$ of the neighborhood. To be invariant to orientation, common descriptor methods anchor a reference coordinate frame directionally parallel to the averaged main gradient orientation of the pixels in the elliptical neighborhood. The gradient orientation coarsely captures the average gradient direction of the local image structure.

We now construct *edges* between these two image features. Using the meta data provided by the feature extractor, we derive four *edge attributes* that describe the relationship between the two features. The following edge attributes are produced.

- $\log\left(\frac{\text{scale}(v_i)}{\text{scale}(v_j)}\right)$, the *ratio of scales* of the adjacent interest points. The scale approximates the size of the captured structure linearly. The use of the logarithm allows to subtract two such features such that the result corresponds linearly to the quotient of two such features. This allows scale invariant matching of scale ratios through a normal RBF kernel. For many features, the values of this edge feature ranges from 0.25 to 4.0. If only large scale features are kept, the range is reduced.

- $\frac{\text{dist}(v_i,v_j)}{\text{scale}(v_i)}$, the Euclidean distance between the interest points $v_i$ and $v_j$ in the image, normalized with respect to the originating point's scale. In the above figure, $\text{dist}(v_i, v_j) = |C_{i,j}|$. The normalized values in our experiments range from 0.0 to 5.0.

- $\cos(\angle(\vec{g}_i, \vec{g}_j))$, the cosine function evaluated on the angle between the keypoints' orientation vectors. Encoding this angle captures information about the local image structure between keypoints. The values naturally range from 0.0 to 1.0.

- $\cos(s_i - s_j) = \cos(\angle(\vec{e}_i, \vec{e}_j))$, the cosine function evaluated on the angle between the major axis direction of the ellipses. Again, the values are inbetween zero and one.

By only using the relative differences between the meta data, we retain all invariances the original feature detector had. That is, scale and position information is only used relatively between features and the absolute value is never used. This is the main reason why it is difficult to incorporate the meta data information into set based approaches: absolute values destroy desired invariances, but the use of relative values requires the consideration of two or more features at one time.

### 4.2.3 Edge kernel

When two graphs are compared, two attributed edges $e_{i,j}$, $e'_{k,l}$ – one from each graph – are compared with each other through an *edge kernel* $K_e$. The edge kernel should produce a high value if two edges are similar and a low value otherwise. For this we use the *product kernel* from section 2.3.4 to obtain

$$K_e(e_{i,j}, e'_{k,l}) = \prod_{m=1}^{4} K_e^{(m)}(A_{e_{i,j},m}, A'_{e'_{k,l},m}, \sigma_m), \tag{2}$$

where $A_{e_{i,j},m}$ is the $m$'th attribute of the edge $e_{i,j}$, $K_e^{(m)}$ is a subkernel for only this attribute type, and $\sigma_m$ is an additional parameter to this kernel. We choose Gaussian RBF kernels for all subkernels. In chapter 5 we will examine how to chose the $\sigma$-parameters.

### 4.2.4 Normalization

The Gram matrix $K$ is normalized such that the kernel values $K_{i,i} = 1$. This is achieved by

$$K'_{i,j} := \frac{K_{i,j}}{\sqrt{K_{i,i}K_{j,j}}}.$$

Compare with section 2.3.4.

### 4.2.5 Explicit path length kernel

To examine the behaviour of the MGK in more detail, we propose another kernel. The MGK (29) computes the marginalized kernel of the form (24)

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} K_z(z, z') p(\mathbf{h}|\mathbf{x}) p(\mathbf{h}'|\mathbf{x}'),$$

with $p(\mathbf{h}|\mathbf{x})$ being the path probability. Due to the design of the MGK and to be able to compute the kernel efficiently, $p(\mathbf{h}|\mathbf{x})$ is defined indirectly through the start, transition and termination probabilities. This makes it unavoidable that paths of all possible lengths are considered.

To check whether single-vertice paths whose corresponding term in (29) only uses the vertex kernel, dominate the MGK result, we do a validation experiment and explicitly compute (24) such that only paths of a fixed given length are considered. Most interestingly is the classification performance that results if we only permit paths of length two with no repeating nodes. Comparing two such paths, one from each graph, then always considers the vertices *and* their geometric relationships. Therefore the edge kernel is always used, in every comparison between two paths. Thus, the kernel then becomes

$$K(G, G') = \frac{1}{|G|} \frac{1}{|G|-1} \frac{1}{|G'|} \frac{1}{|G'|-1} \sum_{v_1} \sum_{\substack{v_2 \neq v_1, \\ e_{v_1,v_2} \in E}} \sum_{v'_1} \sum_{\substack{v'_2 \neq v'_1, \\ e_{v'_1,v'_2} \in E'}} K_v(v_1, v'_1) K_e(e_{v_1,v_2}, e_{v'_1,v'_2}) K_v(v_2, v'_2). \tag{3}$$

The function (3) is still a valid *Marginalized Kernel* of the form (24) of section 2.4.3. For the subsequent experiments we will refer to this kernel as *explicit path length kernel*.

### 4.2.6 Summary

The main structure of our approach is given by the Marginalized Graph Kernel: we need to define vertex and edge attributes and specify suitable discrete probability distributions for the random graph walks.

As *vertex attributes* we use the SIFT descriptors extracted from the image. For the *edge attributes* we derive new relational attributes from the available meta data for every possible pair of interest points. The absolute meta data of a single interest point could not be used for the classification task without destroying invariance properties. Our use of derived relative attributes ensures we retain all invariances. In the next chapter we will identify a good edge kernel based on these attributes.

For the *graph structure* – or, equivalently the probability distributions – we use a fully connected, uniform directed graph. Random walks in this graph then relate to hypotheses of subsets of the features and their relationships as encoded by the edge attributes.

## 4.3 Implementation

The local image features are extracted using the binaries from Mikolajczyk[1], which are known to offer state-of-the-art image feature extraction performance. We use both the Hessian-affine and Harris-affine features, and details are found in chapter 5.

The graph construction and evaluation of the MGK is implemented in C#, using the efficient sparse matrix functions of the dnAnalytics Numerical Library[2]. It produces the Gram matrix from the feature extractor data. The Gram matrix $K'$ is used as input to the Spider Matlab toolkit[3] with LibSVM [8] as backend.

[1] `http://www.robots.ox.ac.uk/~vgg/research/affine/`
[2] `http://www.dnanalytics.net/numerical/`
[3] `http://www.kyb.tuebingen.mpg.de/bs/people/spider/`

# Chapter 5

# Experiments and results

In the last chapter we have described our approach to the object classification problem based on the Marginalized Graph Kernel of section 2.4. In this chapter we evaluate the approach and determine good parameters to use with it. The results of the approach are compared with available results of competing approaches from the literature. The results shown in this chapter will then be summarized in the next chapter, concluding this thesis's work.

## 5.1 The ETH-80 database

The ETH-80 database is an established collection of images for assessing the performance of general objects classification systems. The database was produced for the CogVis project[1] and is available online free of charge.[2]

The eight object classes are: apples, cars, cows, cups, dogs, horses, pears and tomatoes. For each object class, there are ten distinct objects. For each of these 80 objects, 41 pictures are made in front of a blue background. The pictures are made from the same 41 relative positions for each object. Together the set contains 3280 images and includes high quality foreground-background segmentation masks for all images. A small selection of the eight classes is shown in figure 5.1.

Although most scenic variances, such as variations in illumination, background and scale are missing in the ETH-80 set, it is considered a difficult benchmark due to the variances of objects within one class. While some objects are very uniform, such as the cups, apples, tomatoes and pears, the remaining classes – cows, cars, dogs and horses – contain widely varying objects, differing in shape and color.

### 5.1.1 ETH-80 subsets

The ETH-80 set is quite large and most kernel based approaches to object classification are computationally expensive. Hence, for the purpose of evaluating different approaches, representative subsets of ETH-80 are used. We use two subsets, one for which there are classification results available in the literature and one very small one for exhaustive parameter selection purposes.

[1] `http://www.mis.informatik.tu-darmstadt.de/Research/Projects/cogvis/index.html`
[2] `http://www.mis.informatik.tu-darmstadt.de/Research/Projects/categorization/eth80-db.html`

Figure 5.1: ETH-80 set classes, from top to bottom: apples, cars, cows, cups, dogs, horses, pears and tomatoes.

Figure 5.2: ETH-80 minimal set example, "car5" with a camera position of $(66°, 63°)$.

**ETH-80 Eichhorn A subset.** This subset of 400 images has been used to evaluate object classification approaches by Eichhorn and Chapelle [13]. It contains images of all 80 objects, each with five images. The five images used show each object in five widely varying viewpoints.[3] The subset is considered difficult by Eichhorn and Chapelle [13].

**ETH-80 minimal set.** This subset contains 80 images, one for each object.[4] While the sparsity of samples makes generalization somewhat more difficult, the camera is positioned as to show the objects from a diagonal view, exposing a lot of relevant details for the complicated object classes. An example picture is shown in figure 5.2.

## 5.2 Local image features

As interest points and descriptors, we use both the Hessian-affine and Harris-affine regions of Mikolajczyk [37], with the original SIFT descriptor of Lowe [33].

Recent results [36] indicate that Hessian-affine detected interest points may provide a better performance for object classification problems than Harris-affine features. We found that the Harris-affine detected points better capture the inherent detail level of the objects: simple objects like apples yield fewer interest points compared to more structured object like cars. We suspect this is not only an issue of adjusting threshold values but of the cornerness measure used; the Harris measure responds less to textured areas, whereas the Hessian measure also detects points within such regions.

We use both descriptor types for most experiments and for each experiment describe which type has been used.

## 5.3 Experiments

To tune and then evaluate our approach, we propose the following four experiments.

[3] The ETH-80 filename suffixes `000-000`, `035-045`, `066-153`, `090-090` and `090-180` are used.
[4] All the images with the `066-063` suffix.

1. Parameter selection for the edge kernel.

   The edge kernel combining relative information of feature pairs is one of the novel ideas in this work. To maximize the impact of using the edge kernel, a parameter selection is performed and a combination is selected that shows good classification performance.

2. Evaluation of the MGK approach on ETH-80 Eichhorn A.

   The proposed approach of chapter 4 is evaluated on the Eichhorn A set for different parameters.

3. Evaluation of the explicit path length kernel.

   To provide other results for comparison and ground for a following discussion, we further evaluate the explicit path length kernel of section 4.2.5 on the Eichhorn A set. It will be interesting to see whether there is a gap in classification performance between the path length kernel and the MGK based kernel, because both use the idea of an edge kernel but they give different weights to paths in the graph.

4. Single feature-feature comparison kernel.

   Our approach is centered on the use of meta information available for the features. As these features are used only at the edges of the graph, we produce a baseline comparison classifier not using this information, by simply removing all edges from the graph.

**Experimental setup.** For all the experiments, the following experimental setup is used. The image set contains the images of 80 distinct objects, each belonging to one of eight object classes. To obtain a validated testing result, the set of images is split into a training set and a testing set. The training set consists of all images except the images of one object. These few images constitue the testing set. A classifier is then trained on the training set and the classification performance is evaluated on the test samples. This process is repeated once for each object, totaling 80 training runs. The mean of the test performances of all runs is then used as overall classification rate of the system. This is called *leave-one-object-out* testing and is the accepted way for evaluating object classification systems because at no time an image is tested where any other image of the same object has been member of the training set.[5]

Each classifier is trained as follows. The classification is an eight class problem so we use the one-vs-one classification system of section 2.3.6 to reduce it to $\sum_{i=1}^{8-1} i = 28$ two-class problems. The multiclass decision is then done by the *OneVsOneVoting* voting algorithm. For each of the 28 two-class problems, one Support Vector Machine is trained with a maximum weight parameter $C = 100$. On average, the overall mean classification performance for a given kernel matrix in our experiments can be obtained in two minutes. This is much shorter than the time it requires to produce the kernel matrix.

### 5.3.1 Finding the optimal edge kernel

The attributes of the edges in the graph are compared through the edge kernel $K_e$, which we defined as the product of Gaussian RBF kernels between the individual attributes. To obtain suitable $\sigma$-values for each attribute, we use the ETH-80 minimal subset as small parameter selection set.

[5] For the case when each object is represented by only one image in the training set, the leave-one-object-out testing becomes the popular leave-one-out testing.

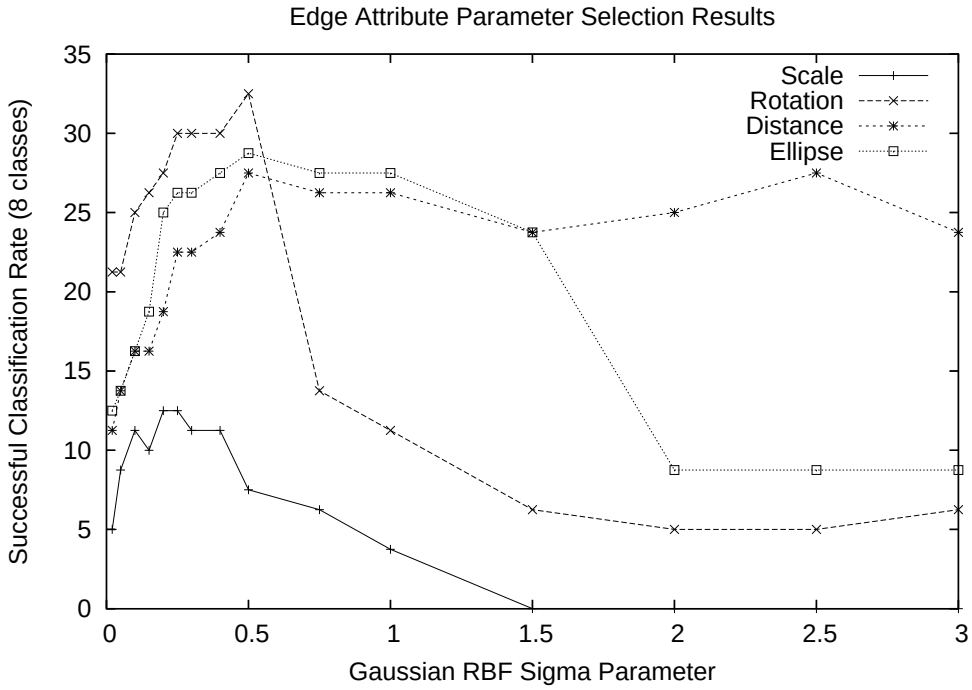


Figure 5.3: Parameter selection for the individual edge attributes on the ETH-80 minimal subset with Harris-affine features.

**Experiment.** We examine both Harris-affine and Hessian-affine extracted features. For each, only the 10 features with the largest scale value are used and $\alpha_s = 1$. The vertex kernel is set to $K_v := 1$ for all nodes and all but one edge attribute is enabled. For this edge attribute the $\sigma$ value is chosen that maximizes the overall leave-one-out verified classification rate.

**Results.** The successful classification rates obtained by using only one edge attribute for different $\sigma$ values are shown in figure 5.3 for Harris-affine features and in figure 5.4 for Hessian-affine features.

Note that for cases where the Gaussian $\sigma$ is very small or too large compared to the attribute values, the discriminatory power is lost and in some cases no class could be assigned due to degenerated kernel values. These sample were then counted as wrongly classified, explaining the classification rates below the expected random guess of 12.5%.

**Chosen parameters.** The edge kernel parameters are chosen separately for Harris-affine and Hessian-affine features. For Harris-affine features we use $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\sigma_3 = 0.75$ and $\sigma_4 = 0.5$ for the scale, rotation, distance and ellipse attributes, respectively. For Hessian-affine features we use $\sigma_1 = 0.05$, $\sigma_2 = 0.02$, $\sigma_3 = 1.0$ and $\sigma_4 = 0.1$ for the respective scale, rotation, distance and ellipse attributes.

To further select the optimal combination of features, all 15 possible product combinations between the individual attribute kernels are considered. The results are described in the following two paragraphs.

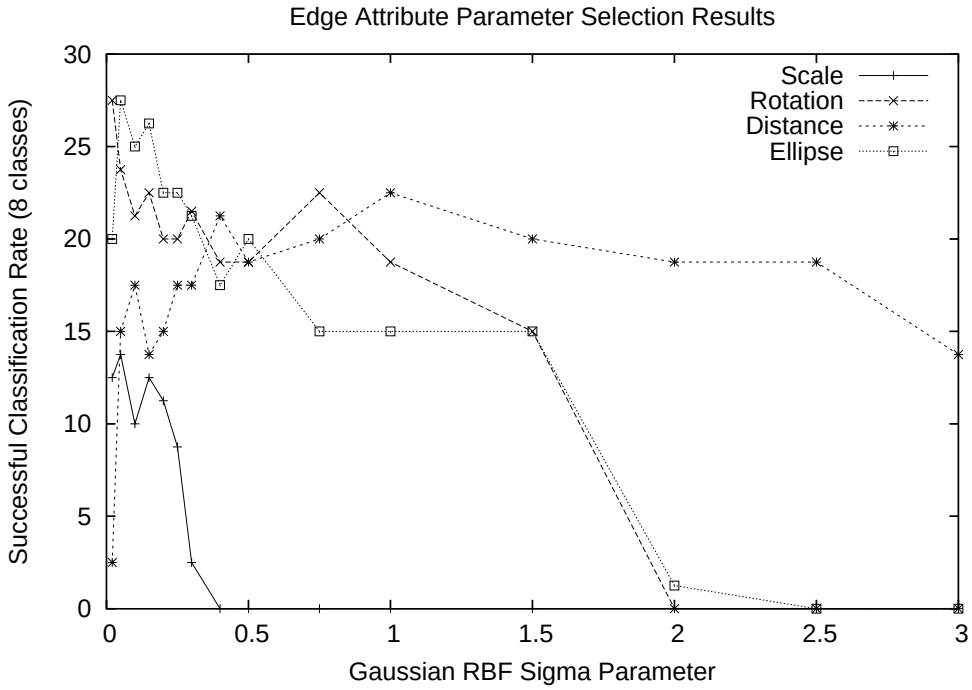


Figure 5.4: Parameter selection for the individual edge attributes on the ETH-80 minimal subset with Hessian-affine features.

**Harris-affine features.** Consistently, product combination "A" (scale, distance, ellipse) and product combination "B" (scale, rotation, ellipse) resulted in the best classification rate. On the same setup as before we obtain a 35.0% classification rate for both combinations with a maximum of 10 features per image and $\alpha_s = 1$. To validate this edge kernel, a small test with different number of features and values of $\alpha_s$ is done, the results of which are shown in table 5.1.

**Hessian-affine features.** For this feature type, the strongest classification rate is obtained with the product combination "A" (scale, distance, ellipse) with a classification rate of 38.75% and combination "C" (distance, ellipse) with a rate of 43.75%, each with $\alpha_s = 1$.

Table 5.1: Edge kernel classification results on the ETH-80 minimal subset with the combined product kernel. Harris-affine features with a maximum of 10, 15 and 20 points are used.

| Kernel | max features | $\alpha_s = 0$ | $\alpha_s = 1$ | $\alpha_s = 2$ | $\alpha_s = 3$ | $\alpha_s = 4$ | $\alpha_s = 5$ |
|---|---|---|---|---|---|---|---|
| A | 10 | **35.0** | **35.0** | 32.5 | 30.0 | 31.25 | 32.5 |
| A | 15 | 33.75 | 42.5 | **45.0** | **45.0** | 40.0 | 35.0 |
| A | 20 | 37.5 | 35.0 | 36.25 | 40.0 | 36.25 | **38.75** |
| B | 10 | **41.25** | 35.0 | 36.25 | 37.5 | 32.5 | 28.75 |
| B | 15 | 31.25 | 40.0 | **42.5** | 41.25 | 38.75 | 41.25 |
| B | 20 | 37.5 | 36.25 | 40.0 | 47.5 | 46.25 | **51.25** |

Table 5.2: Edge kernel classification results on the ETH-80 minimal subset with the combined product kernel. Hessian-affine features with a maximum of 10, 15 and 20 points are used.

| Kernel | max features | $\alpha_s = 0$ | $\alpha_s = 1$ | $\alpha_s = 2$ | $\alpha_s = 3$ | $\alpha_s = 4$ | $\alpha_s = 5$ |
|---|---|---|---|---|---|---|---|
| A | 10 | 38.75 | 38.75 | **40.0** | 38.75 | 36.25 | 38.75 |
| A | 15 | 25.0 | 31.25 | 28.75 | 31.25 | **32.5** | **32.5** |
| A | 20 | 17.5 | 16.25 | 17.5 | 18.75 | 20.0 | **23.75** |
| C | 10 | **43.75** | **43.75** | 40.0 | 36.25 | 36.25 | 41.25 |
| C | 15 | **45.0** | 43.75 | **45.0** | 42.5 | 40.0 | 38.75 |
| C | 20 | 46.25 | 43.75 | 45.0 | **47.5** | **47.5** | **47.5** |

The same parameter variation experiment as for the Harris-affine features is repeated and the results is shown in table 5.2. The edge kernel "A" does not show good results and also the classification rate does not increase with more available features. For the remaining experiments with Hessian-affine features and active edge kernel, we hence use combination "C".

**Testing on a larger set.** To assess the qualitative strength and weaknesses of the edge kernel, we perform an experiment on the larger and more difficult ETH-80 Eichhorn A subset. The A kernel is used with both Hessian-affine and Harris-affine extracted features. The results of the Harris-affine features are shown in table 5.3, and the results of the Hessian-affine features are shown in table 5.4.

For the Harris-affine features, the classification rate on the larger and more difficult Eichhorn A set is disappointing at 22.75%. The decrease relative to the ETH-80 minimal set results can be explained by the additional images in the Eichhorn A set which do not contain as much information per image. The added variation in pose makes the classification more difficult. As the available information per image decreases, there is fewer relevant information to make a class decision. In the confusion matrix, this leads to a diffusion over the predicted class rows, biased towards "similar" object classes: apples become tomatoes, cows become dogs, horses become cars, etc.

For the Hessian-affine features, the classification performance is 27.0% and the confusion matrix is quite similar to the Harris-affine case, although the situation has improved somewhat. The strongest confusions are between cars and cows, cows and horses, and apples and tomatoes. So confusions happen for both simple objects, such as apples and tomatoes, as well as for structured objects, such as cars, cows and horses.

**Conclusion.** Clearly the edge kernel does capture some relevant information for object classification. As only very few features and information about each feature is used, the results are well below a good classification rate. The point of the experiment is to prove the relevance, not the superiority of the edge kernel. It is not easy to introspect the workings of the edge kernel directly because it is used within the context of the MGK. But here we give two possible explanations for why the edge kernel works. Each explanation takes a simplified and relatively extreme view; the real explanation may just be in between.

1. *Explicit feature correspondence view.* While objects vary in one class, it is likely that two similar objects in one class have similar interest points detected at similar places.

Table 5.3: The confusion matrix of the edge kernel A evaluated on the ETH-80 Eichhorn A data set. Here $\alpha_s = 1.0$ and a maximum of 10 Harris-affine features per image have been used. The total classification rate is 22.75%.

| | *predicted classes* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *actual classes* | apple | car | cow | cup | dog | horse | pear | tomato |
| apple | **16** | 1 | 3 | 6 | 6 | 5 | 2 | 11 |
| car | 0 | **15** | 7 | 4 | 3 | 11 | 5 | 5 |
| cow | 2 | 10 | 4 | 8 | **14** | 4 | 6 | 2 |
| cup | 1 | 7 | 6 | **15** | 3 | 1 | 6 | 11 |
| dog | 3 | 1 | 6 | 5 | 8 | **12** | 8 | 7 |
| horse | 2 | **12** | 5 | 4 | 10 | 8 | 4 | 5 |
| pear | 5 | 4 | 5 | 8 | 5 | 4 | **14** | 5 |
| tomato | 4 | 7 | 5 | 6 | 6 | 6 | 5 | **11** |

Table 5.4: The confusion matrix of the edge kernel C evaluated on the ETH-80 Eichhorn A data set. Here $\alpha_s = 3.0$ and a maximum of 10 Hessian-affine features per image have been used. The total classification rate is 27.0%.

| | *predicted classes* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *actual classes* | apple | car | cow | cup | dog | horse | pear | tomato |
| apple | **18** | 2 | 2 | 7 | 4 | 2 | 0 | 15 |
| car | 2 | **16** | **16** | 4 | 2 | 5 | 0 | 5 |
| cow | 3 | **13** | 3 | 0 | 9 | 9 | 6 | 7 |
| cup | 10 | 3 | 2 | **22** | 2 | 2 | 5 | 4 |
| dog | 3 | 2 | 5 | 1 | **12** | 8 | 11 | 8 |
| horse | 1 | 8 | **13** | 2 | 10 | 9 | 2 | 5 |
| pear | 1 | 3 | 3 | 5 | 10 | 5 | **19** | 4 |
| tomato | **10** | 5 | 3 | 9 | 4 | 8 | 2 | 9 |

The edge attributes encode the relations between features. Hence, if there are a small number of similar features at similar places, all edges between such features in the graph for one image are similar to the same edges in the other graph for the other image. When the edge kernel compares all edges of the graphs with each other, the non matching edges produce numerical "noise", but the relevant matching edges produce high kernel values and dominate this noise, leading to high kernel values. Thus, the performance is the result of a few strong matches among edges. Given the class is represented by enough samples, there will be two objects similar enough to each other.

2. *Sampling the feature relation statistics view.* The in-class variation in shape and color of the objects is so significant that the local image features are unlikely to appear at similar places with similar characteristics. Still, the distribution and relations of the features in the image contains relevant information: the features may not be the same or appear at similar places, but the way they relate to each other may be similar. Evaluating the edge kernel for all possible path matches can then be seen as sampling the intersection of the distributions of edge attribute values with some intersection measure defined implicitly by the edge kernel. Object within the same class are assumed to have a larger intersection in these distributions than objects in two different classes. Thus, the performance is the result of a broad statistical correspondence between edge attributes of the objects of one class.

We have found evidence for both claims, and the bias towards one explanation is different for each class. The kernel matrix shown in figure 5.5 illustrates this situation. There are more well isolated classes like cars (elements 21-30) and classes with a broad spread of high kernel values, such as apples, pears and tomatoes (elements 1-10, 61-70 and 71-80, respectively). We suggest that for the latter classes the second explanation might be the case, and for the highly structured classes the first explanation is more likely.

### 5.3.2 ETH-80 Eichhorn A

We now evaluate the MGK approach of chapter 4. In general, the parameters used are all as described in chapter 4 and section 5.3.1, except where explicitly given otherwise.

**Experiment.** The MGK approach is evaluated for both Harris-affine and Hessian-affine features on the ETH-80 Eichhorn A set. Different values for $f_\sigma$ and $\alpha_s$ are tested and we use the edge kernel "A" for Harris-affine features and the combination "C" for Hessian-affine features. The results are shown in table 5.5.

The confusion matrix of the MGK classification system for Harris-affine features is shown in table 5.6.

**Discussion.** From table 5.5 it is clear that Hessian-affine features outperform Harris-affine ones by an additional five percent. As could be expected when the same feature descriptor method is used, the same values for $f_\sigma$ produce the maximum classification performance. For both feature types, $\alpha_s = 3$ leads consistently to better performance than $\alpha_s = 5$.

These results will be compared to the other experiments in section 5.4.1.

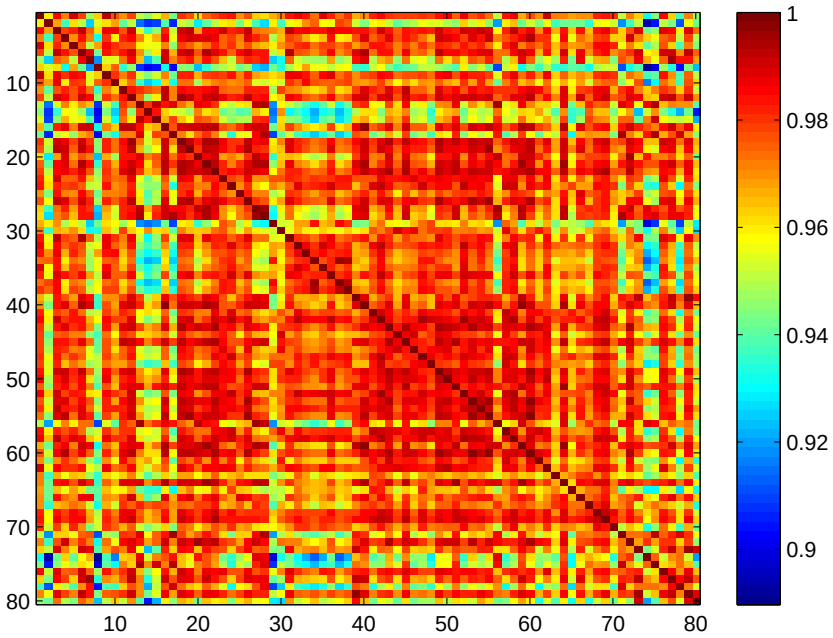


Figure 5.5: Kernel matrix plot of the edge kernel C on the ETH-80 minimal subset. A maximum of 10 Hessian-affine features are used.

Table 5.5: ETH-80 Eichhorn A set classification results of our MGK based approach with both Harris-affine and Hessian-affine SIFT features. A maximum of 10, 15 and 20 features is used per image and the feature sigma $f_\sigma$ is varied. The edge kernel parameters are chosen based on the experiment in section 5.3.1 and are not the same for the Harris-affine and Hessian-affine features.

| | | Classification results | | | |
|---|---|---|---|---|---|
| | maximum no. | *Harris-affine* | | *Hessian-affine* | |
| $f_\sigma$ | of features | $\alpha_s = 3$ | $\alpha_s = 5$ | $\alpha_s = 3$ | $\alpha_s = 5$ |
| 100 | 10 | 43.75 | 41.5 | 48.0 | 44.0 |
| 100 | 15 | 45.25 | 41.75 | 51.0 | 45.25 |
| 100 | 20 | 45.25 | 42.0 | 51.75 | 45.5 |
| 200 | 10 | 57.5 | 54.75 | 69.0 | 65.75 |
| 200 | 15 | 59.0 | 55.25 | 70.5 | 66.0 |
| 200 | 20 | 59.75 | 55.75 | 71.25 | 66.5 |
| 250 | 10 | 59.0 | 57.0 | 67.0 | 64.5 |
| 250 | 15 | 60.75 | 58.75 | 70.25 | 65.25 |
| 250 | 20 | 61.25 | 61.0 | 71.0 | 66.5 |
| 300 | 10 | 56.75 | 55.5 | 66.25 | 63.5 |
| 300 | 15 | 59.75 | 56.5 | 69.5 | 64.25 |
| 300 | 20 | 61.0 | 56.0 | 71.5 | 65.25 |
| 500 | 10 | 54.0 | 52.25 | 65.0 | 61.75 |
| 500 | 15 | 56.25 | 54.25 | 67.5 | 64.25 |
| 500 | 20 | 57.5 | 54.0 | 68.25 | 65.5 |

Table 5.6: The confusion matrix of the MGK classifier evaluated on the ETH-80 Eichhorn A data set. Here $f_\sigma = 250$, $\alpha_s = 3.0$ and 20 Harris-affine features per image have been used with the edge kernel "A".

| | *predicted classes* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *actual classes* | apple | car | cow | cup | dog | horse | pear | tomato |
| apple | **39** | 0 | 0 | 3 | 1 | 0 | 1 | 6 |
| car | 0 | **39** | 6 | 1 | 3 | 1 | 0 | 0 |
| cow | 0 | 2 | 16 | 0 | **18** | 13 | 0 | 1 |
| cup | 3 | 2 | 0 | **34** | 1 | 7 | 1 | 2 |
| dog | 0 | 4 | 13 | 2 | **16** | 15 | 0 | 0 |
| horse | 0 | 4 | 13 | 2 | 12 | **18** | 1 | 0 |
| pear | 0 | 0 | 0 | 1 | 1 | 2 | **44** | 2 |
| tomato | 6 | 1 | 1 | 0 | 0 | 0 | 3 | **39** |

### 5.3.3 Explicit path length kernel

We evaluate the explicit path length kernel of section 4.2.5 on the Eichhorn A data set with a path length of two and Hessian-affine features. As edge kernel, the optimal configuration determined in section 5.3.1 is used. Two parameters remain: the feature $\sigma$ and the sub-polynomial kernel exponent $E$. To determine a good combination, a simple grid parameter selection is done on $\sigma \in \{50, 100, 150, 200, 250, 300, 400, 500, 600, 1000\}$ and $E \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. For efficiency reasons, the number of keypoints is pruned to a maximum 10. Because in this experiment, the edge kernel is always used for each comparison happening in the kernel, at least two features are needed per image.

**Results.** The result shown in table 5.7 is visualized in the surface plot 5.6. A good and stable configuration seems to be $\sigma = 200$, $E = 0.6$. Considering only ten features are used per image, the classification performance of 69.75% for eight classes is very good. The confusion matrix of the classifier with this parameters is shown in table 5.8. There are no surprises in the confusion matrix and all the confusions are quite natural: the biggest mutual confusions are between dogs and horses and apples and tomatos. The biggest misclassification in one direction is for cows, where 12 of 50 samples are classified as dogs. The largest receivers of misclassifications – having many non-zero values in their respective columns – are highly structured object classes: cars, cows, dogs and horses. The largest confusion happens within the set of cows, dogs and horses, while all other classes are more separated. In my subjective judgement the images of cows, dogs and horses within ETH-80 are quite similar.

To get an impression how the feature space is formed, we obtain a low dimensional approximation to the space as follows.[6] The kernel matrix is converted to a distance matrix between samples as described in section 2.3.4. The distance matrix is then used as input to Multi-dimensional Scaling (MDS) [12] to obtain coordinates for each sample in a low dimensional space. MDS choses the the coordinates such that the minimum linear residue is obtained for the mapped points' distances to each other. Hence, if the residue becomes small enough a good approximation to the space has been obtained. The points are then drawn in a scatter

[6]The real feature space is likely to have as many dimensions as there are samples.

Table 5.7: Grid selection results for the explicit path length kernel and the feature $f_\sigma$ and exponent $E$ parameters. A maximum of ten Hessian-affine extracted features are used per image.

| | *feature* $f_\sigma$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *exponent* $E$ | 50 | 100 | 150 | 200 | 250 | 300 | 400 | 500 | 600 | 1000 |
| 1.0 | 1.00 | 20.25 | 48.50 | 64.00 | **68.00** | 67.00 | 63.50 | 58.75 | 55.25 | 44.50 |
| 0.9 | 1.50 | 24.00 | 53.25 | 65.75 | **67.50** | 65.75 | 63.25 | 58.75 | 55.25 | 45.75 |
| 0.8 | 3.00 | 29.25 | 58.25 | 67.00 | **68.00** | 65.25 | 62.75 | 57.75 | 55.25 | 46.50 |
| 0.7 | 4.50 | 34.25 | 62.00 | **68.00** | 67.50 | 65.50 | 61.75 | 57.50 | 55.25 | 46.25 |
| 0.6 | 5.25 | 40.25 | 65.00 | **69.75** | 66.75 | 64.50 | 60.50 | 57.25 | 54.75 | 44.50 |
| 0.5 | 8.75 | 49.25 | 67.50 | **69.00** | 65.50 | 64.25 | 61.00 | 58.00 | 54.00 | 44.00 |
| 0.4 | 16.25 | 61.00 | 68.50 | **68.75** | 65.75 | 64.00 | 60.75 | 57.25 | 53.75 | 43.25 |
| 0.3 | 27.75 | 68.00 | **70.25** | 67.50 | 65.25 | 63.75 | 61.75 | 57.25 | 54.25 | 38.50 |
| 0.2 | 40.25 | **70.50** | 69.00 | 67.00 | 66.75 | 63.00 | 60.75 | 55.50 | 52.25 | 37.50 |
| 0.1 | 67.75 | **71.00** | 68.25 | 66.00 | 65.25 | 63.25 | 58.75 | 52.50 | 46.25 | 35.25 |

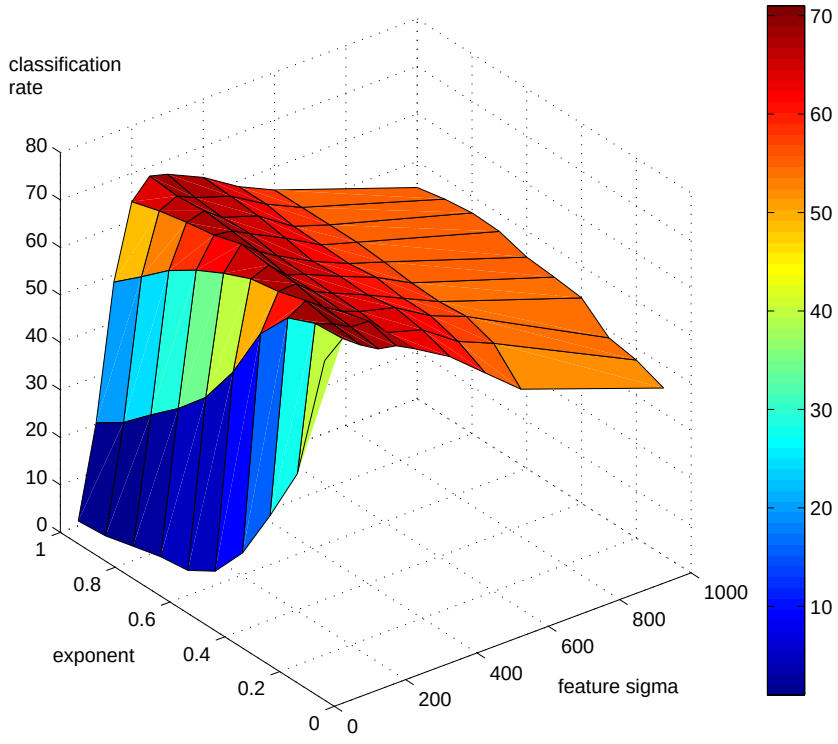


Figure 5.6: Surface plot of the grid parameter selection results. The clear ridge shows a maximum in the direction of the feature sigma, corresponding to the bold-marked results in table 5.7.

Table 5.8: Confusion matrix for the explicit path length kernel experiment. A maximum of 10 Hessian-affine features are used, $f_\sigma = 200$, and edge kernel "C" is used. The classification rate is 69.75%.

| | *predicted classes* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *actual classes* | apple | car | cow | cup | dog | horse | pear | tomato |
| apple | 43 | 0 | 0 | 0 | 1 | 0 | 1 | 5 |
| car | 0 | 44 | 2 | 0 | 3 | 1 | 0 | 0 |
| cow | 0 | 1 | 26 | 1 | 12 | 10 | 0 | 0 |
| cup | 0 | 4 | 0 | 39 | 4 | 2 | 1 | 0 |
| dog | 0 | 2 | 6 | 0 | 23 | 19 | 0 | 0 |
| horse | 0 | 4 | 6 | 0 | 19 | 20 | 1 | 0 |
| pear | 0 | 0 | 0 | 1 | 5 | 0 | 44 | 0 |
| tomato | 9 | 0 | 1 | 0 | 0 | 0 | 0 | 40 |

Table 5.9: Grid selection results for the single feature-feature comparison kernel on the ETH-80 Eichhorn A set, $f_\sigma$ is varied. A maximum of ten, 15 and 20 Hessian-affine extracted features are used per image.

| | *feature* $f_\sigma$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *max. features* | 50 | 100 | 150 | 200 | 250 | 300 | 400 | 500 | 600 | 1000 |
| 10 | 11.75 | 48.75 | 67.0 | **69.0** | 67.5 | 65.0 | 64.25 | 63.5 | 62.5 | 57.25 |
| 15 | 13.25 | 51.5 | 67.5 | **70.5** | 69.0 | 68.75 | 66.75 | 65.25 | 64.25 | 60.5 |
| 20 | 13.75 | 52.0 | 68.75 | 71.0 | 70.5 | **71.25** | 68.75 | 67.25 | 66.5 | 60.0 |

plot. We show one good classifier's output in figure 5.7. Though the depth information is difficult to guess from the figure, one can see the clear overall separation of some classes and the nearby mapping of related classes.

### 5.3.4 Single feature-feature comparison kernel

A straightforward set kernel can be derived by removing all edges from the graph and applying the MGK between such degenerate graphs. This produces surprisingly good results, as shown in table 5.9. To our surprise, the results are on-par with the explicit path length kernel and the MGK based approaches. We will discuss possible reasons for this in the next section.

## 5.4 Discussion

In the experiments so far we have both examined and tuned individual parts of the system, as well as evaluated the overall MGK based classification system. What do the results so far tell us?

First, three approaches are equal in their classification rate: the Hessian-affine features with either the MGK, the explicit path length kernel or the single feature-feature kernel all produce around 70% validated successful classifications on the ETH-80 Eichhorn A dataset.

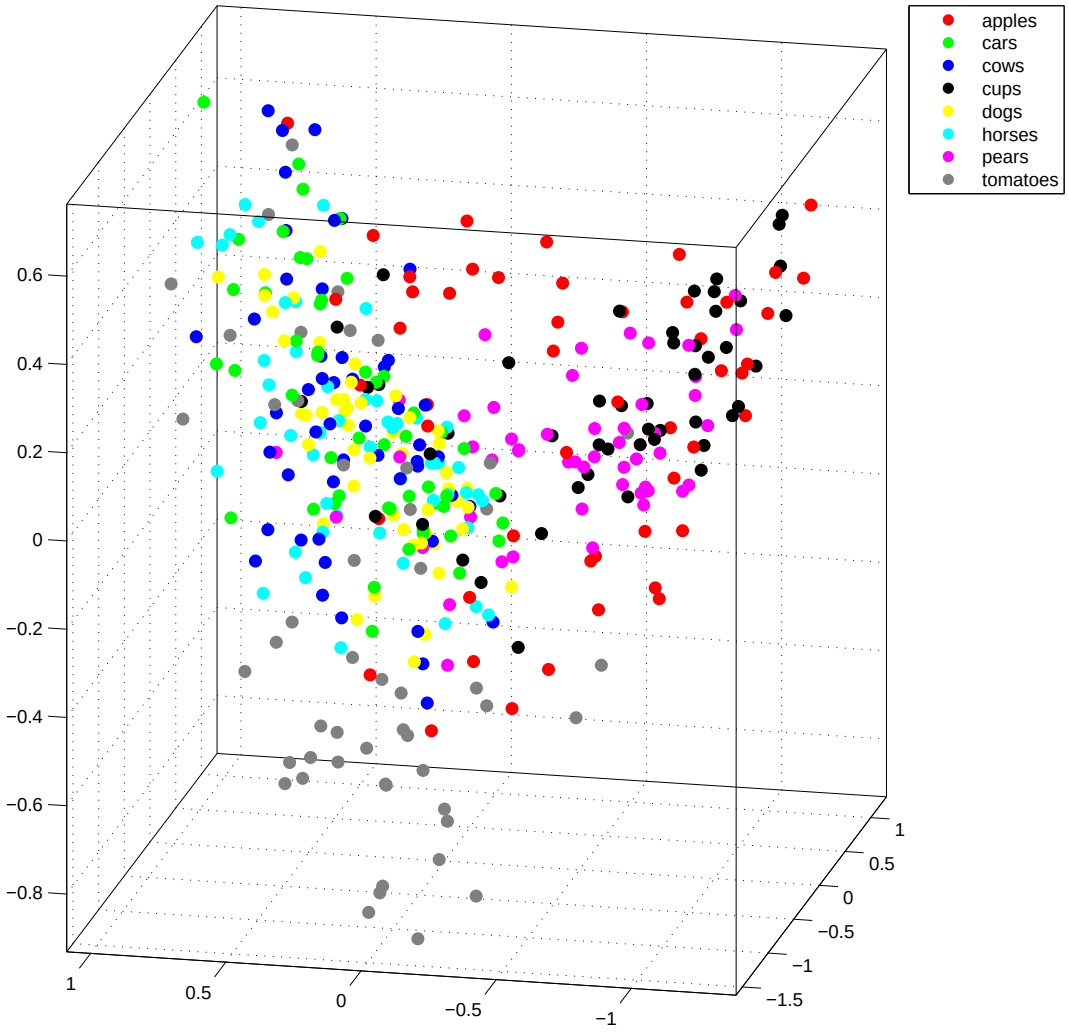


Figure 5.7: Feature space approximation scatter plot of the explicit path length kernel output. The kernel matrix is converted into a distance matrix, which is then mapped using Multidimensional Scaling (MDS) to produce 3D coordinates. The residual variances after 1, 2 and 3 dimensions were 0.47455, 0.14105 and 0.089869; the overall structure of the feature space is well captured. The parameters for the explicit path length kernel were $f_\sigma = 200$, 10 Hessian-affine features per image, pathlength 2, subpolynomial kernel exponent $E = 0.6$.

Table 5.10: ETH-80 Eichhorn A set classification results. Results from A: [13], B: [20].

| Method | Feature | Source | Avg. feat. | Result |
|---|---|---|---|---|
| Matching kernel [54] | SIFT, NormCC | A | 40 | 74% |
| Bhattacharyya [27] | SIFT, RBF Gaussian | A | 40 | 74% |
| Bhattacharyya [27] | SIFT, RBF Gaussian | A | 120 | 86% |
| Pyramid Match [20] | Harris-aff. PCA-SIFT | B | 40 | 68% |
| Pyramid Match [20] | Harris-aff. PCA-SIFT | B | 153 | 82% |
| MGK | Harris-aff. SIFT, $\alpha_s = 5$, $f_\sigma = 250$ | | max 10 | 57.0% |
| MGK | Harris-aff. SIFT, $\alpha_s = 5$, $f_\sigma = 250$ | | max 15 | 58.75% |
| MGK | Harris-aff. SIFT, $\alpha_s = 5$, $f_\sigma = 250$ | | max 20 | 61.0% |
| MGK | Hessian-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 200$ | | max 10 | 69.0% |
| MGK | Hessian-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 250$ | | max 15 | 70.25% |
| MGK | Hessian-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 300$ | | max 20 | 71.5% |
| explicit path length | Hessian-aff. SIFT, $\alpha_s = 0$, $f_\sigma = 200$, $E = 0.6$ | | max 10 | 69.75% |
| single features | Harris-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 250$ | | max 10 | 58.75% |
| single features | Harris-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 250$ | | max 15 | 60.25% |
| single features | Harris-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 250$ | | max 20 | 61.25% |
| single features | Hessian-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 250$ | | max 10 | 67.5% |
| single features | Hessian-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 250$ | | max 15 | 69.0% |
| single features | Hessian-aff. SIFT, $\alpha_s = 3$, $f_\sigma = 250$ | | max 20 | 70.5% |

Second, for the bad classifications, they are almost always classified into a "similar" classes: cows are taken for horses or tomatoes for apples. This shows that the feature space captures important characteristics of object classes.

With these two observations in mind, we now compare our approach against the literature.

### 5.4.1 Comparison

We now give overall comparison of the results of our different approaches and the results of approaches from the literature. Table 5.10 shows the results. To make a fair comparison we selected the parameters for each one of our approaches that has produced the best results.

### 5.4.2 Main results

1. The proposed edge kernel $K_e$ does have significant discriminative power for the task of object classification. While the edge kernel classification performance alone cannot yet compare well to performance of the other approaches based on local image features, it is a first step to use the meta data – position, size and orientation – of the features in a kernel based classification system. The only other approach in this direction we are aware of is Lyu [34].

2. Surprisingly, the integration of the edge kernel $K_e$ into the Marginalized Graph Kernel does not significantly improve the classification performance over a baseline comparison kernel which removes all edges from the graph.

3. Only considering paths of length two in the explicit path length kernel produces equal classification performance to both the MGK and the single feature-feature kernel.

4. The Hessian-affine features outperform the Harris-affine ones, but the Harris-affine features have the advantage of being sparse for low structure objects, such as the apples in the ETH-80 set, leading to a lower number of features for these classes.

# Chapter 6

# Conclusions

Reaching the end of this work, we now conclude by reviewing our achievements and taking a look at possible future work.

*Summarizing*, we have proposed a novel extension to modern object classification approaches. A graph representation of local image features for object classification was proposed for use with the Marginalized Graph Kernel. The approach has been evaluated on a standard data set.

The *strengths* of the approach are two-fold. First, we demonstrated the relevancy of the additional information. Second, using the graph structure with a graph kernel gives the approach more flexibility than other kernel based object classification systems have.

The *weakness* of the approach is the missing significant increase in classification performance compared to other approaches and the excessive computational demand our approach requires. We discuss the first problem in an extra section.

## 6.1 Weaknesses

While it is apparent that incorporating feature meta data introduces important geometric and global information about an object into the representation, the proposed approach did not show sufficiently increased classification rates.

This basically can have two reasons, i) the information could be redundant for the task at hand or have its value be dominated by other available information, or ii) the information is not properly used by the system.

Under the above two considerations, we identify the following critical points that would deserve further detailed examination.

- The meta information along the edges does not improve the classification rate over SIFT.

  While we clearly demonstrated that the edges contain relevant information for the classification task, it may be the case that the information at the interest points dominates the edge information. That is, in case the information at the interest points – the SIFT descriptors – already contain a lot of relevant information and both information types work well on the same types of objects, then there is no synnergy effect between these two information types by *combining* them.[1]

[1] Clearly the edge information is not *redundant* in the information theory sense, but the edge information

- The influence of longer pathes occuring in both graphs is not properly accounted for by the MGK.

  A long matching path should produce a large kernel value for its discriminative power. In the MGK, such pathes actually receive a smaller weight than even a one-element path. This is the case because $K_v \leq 1$, $K_e \leq 1$ is required for uniqueness of the solution of the MGK. As the final kernel value is a product from these kernel values, a longer path is likely to produce smaller values.

  It is not obvious how to include a bias to favor longer matching pathes over short ones while still retaining a valid kernel function.

We have evaluated two baseline experiments – the single feature-feature kernel and the explicit path lengths kernel – in comparison with the MGK based approach. Considering the results, we tend to favor the first explanation.

## 6.2 Goals

In section 1.2 we have described the goals we want to achieve with this work. Now we quickly revisit the goals and decide whether we have succeeded or not.

We

- gave an overview of current kernel based approaches to object classification in chapter 3.
- identified common shortcomings of these approaches in section 3.5.
- proposed an extended approach to overcome these deficits in chapter 4.
- evaluated and discussed the approach in chapter 5.
- summarized our results in this chapter.

While our approach works, it is not significantly better than current approaches and can hence be considered a failure in improving over current approaches. However, we hope our approach contains some novel ideas, providing the basis for more successful future work.

## 6.3 Future work

A more efficient calculation of the MGK or an alternative graph kernel is needed to improve the usability of the approach. This would also allow more features per image to be used, which has been shown to increase classification performance in most approaches.

Other sources of information should be examined, as the flexibility of a graph allows the incorporation of such information. Candidates for consideration are the scale-space structure the features live in, possibly imposing a hierarchy of features.

would then be *redundant for the classification task*.

# Appendix A

# Appendix A - Source code

## A.1 Educational examples

- `simplesvm.m`

  The `simplesvm.m` contains a minimal hard margin SVM implementation using the `quadprog` solver of Matlab.

```
function [w,b,alphas,M] = simplesvm (X, Y);

% A simple linear hard margin SVM.
%
% Date: 10th February, 2006
% Author: Sebastian Nowozin <nowozin@cs.tu-berlin.de>
%
% Input:
%    X: m n-dimensional points as (m,n) matrix.
%    Y: labels -1 or 1, as (m,1) vector.
%
% Output:
%    w: The separating hyperplane parameter w.
%    b: The bias b.
%    alphas: The Lagrangian multipliers.
%    M: The width of the margin.

% Build the H matrix for the Matlab quadprog function. X*X' is the kernel
% matrix.
H = Y*Y'.*(X*X');
f = -ones(size(Y));

% Constraint: sum of alphas must be zero.
Aeq = Y';
beq = 0;

% Lower bounds: Lagrangian multipliers must be >= 0
```

```
LB = zeros(size(Y));

% Obtain optimal solution.
alphas = quadprog (H, f, [], [], Aeq, beq, LB, []);

w = sum ((alphas.*Y) * ones(1,size(X,2)) .* X)';

% Find a support vector from the positive set.
s = find ((alphas.*Y) > 0.01);
b = Y(s) - X(s,:)*w;

M = 2 / norm(w);
```

- `scalespace.m`

  The file `scalespace.m` creates a Gaussian scale-space approximation like the one used in the SIFT algorithm from section 2.2.4. It does not use subsample fitting and produces no SIFT descriptor, but illustrates the structure of both the scale-space and the Difference-of-Gaussian space.

```
% SIFT scalespace demo
%
% Author: Sebastian Nowozin <nowozin@cs.tu-berlin.de>
% Date: 8th March 2006
%
% Simple Gaussian scalespace approximation with DoG based feature
% localization.  Shows some concepts of the SIFT keypoint localization.

I = double(imread ('sunflower-small.png'));
I = I./(max(max(I)));    % Normalize

dogThresh = 0.02;    % Threshold on LoG filter response
curvatureRatioThresh = 10;    % Ratio of principal curvatures.
count = 13;        % Number of pictures.

s = 3;    % Number of planes per octave
k = 2^(1/s);    % Constant factor between each plane's Gaussian sigma

i = 1;

% Produce scale space and DoG's.
p{1} = I;
for i=1:(count+1)
    convsigma = k^(i-1);
    h = fspecial('gaussian',2*ceil(convsigma*3)+1,convsigma);
```

```
    i = i + 1;
    p{i} = double(imfilter(I,h,'replicate'));
    dog{i-1} = double(p{i}-p{i-1});
    dogsigma{i-1} = convsigma;
end

% The shift masks to reach the DoG plane neighbors.
neighbors = [-1 -1 ; -1 0 ; -1 1 ; 0 -1 ; 0 1 ; 1 -1 ; 1 0 ; 1 1];
planes = [0 -1 1];

% Produce maxima locations.
for n = 2:(length(dog)-1)
    % Index mask.
    Cmax{n} = (abs(dog{n}) > dogThresh) & (dog{n} > dog{n+1}) & (dog{n} > dog{n-1});
    Cmin{n} = (abs(dog{n}) > dogThresh) & (dog{n} < dog{n+1}) & (dog{n} < dog{n-1});

    % Check the remaining 25-neighborhood
    for i=1:length(planes)
        for nei=1:size(neighbors,1)
            Cmax{n} = Cmax{n} & (dog{n} > ...
                circshift(dog{n+planes(i)}, neighbors(nei,:)));
            Cmin{n} = Cmin{n} & (dog{n} < ...
                circshift(dog{n+planes(i)}, neighbors(nei,:)));
        end
    end
    C{n} = Cmax{n} | Cmin{n};

    % Remove border
    C{n}(1,:) = 0;
    C{n}(end,:) = 0;
    C{n}(:,1) = 0;
    C{n}(:,end) = 0;

    % Check the ratio of principal curvature for each keypoint.
    [rows,cols]=find(C{n});
    cr = (curvatureRatioThresh+1)^2 / curvatureRatioThresh;
    for j=1:length(rows)
        r = rows(j);
        c = cols(j);

        % Second order accurate finite differencing schemes.
        D_xx = dog{n}(r,c-1) - 2*dog{n}(r,c) + dog{n}(r,c+1);
        D_yy = dog{n}(r-1,c) - 2*dog{n}(r,c) + dog{n}(r+1,c);
        D_xy = 0.25 * (dog{n}(r+1,c+1) - dog{n}(r-1,c+1) - ...
            dog{n}(r+1,c-1) - dog{n}(r-1,c-1));

        trH = D_xx + D_yy;
```

```
            detH = D_xx*D_yy - D_xy^2;
            if (trH^2 / detH) > cr
                C{n}(r,c) = 0;    % Prune keypoint.
                disp(['Keypoint at plane ', num2str(n), ' at (', ...
                    num2str(c), ',', num2str(r), ') pruned.']);
            end
        end
    end

    % Plot
    iptsetpref('ImshowBorder','tight');
    fig_img=figure;
    for s=2:length(C)
        subplot(3,4,s-1);

        % The image figure
        imagesc(p{s}); colormap(gray);
        axis image;
        %alpha(.5);    % XXX: alpha does not work well with EPS export...

        hold on;
        [r,c]=find(C{s});
        plot(c,r,'ro','MarkerSize',3*dogsigma{s});
        title(['Scale ', num2str(dogsigma{s})]);
        hold off;
    end

    fig_dog=figure;
    for s=2:length(C)
        subplot(3,4,s-1);

        % The DoG figure.
        imagesc(dog{s}); colormap(gray);
        axis image;

        hold on;
        [r,c]=find(C{s});
        plot(c,r,'ro','MarkerSize',3*dogsigma{s});
        title(['Scale ', num2str(dogsigma{s})]);
        hold off;
    end
```

## A.2 Code

- mgk.m

`mgk.m` contains an efficient Marginalized Graph Kernel implementation for small to medium sized graphs with sparse connectivity (10-100 nodes, less than 10 edges per node).

```
function [K] = mgk (g1, g2, K_vertex, K_edge, ...
    prob_T, prob_Q, prob_S, method);

% Marginalized Graph Kernel.
%
% Author: Sebastian Nowozin <nowozin@cs.tu-berlin.de>
% Date: 22nd February 2006
%
% Description: implements the Marginalized Graph Kernel [Kashima2003].
%
% Input:
%    g1: The first graph as structure with the elements:
%        V: (n,m) matrix of n vertices.  Each vertex has a (1,m) data vector.
%        E: (p,2) directed edge list, each row [i1 i2] denotes an edge from
%           the i1'th vertex to i2'nd vertex.  The list must be ordered and no
%           duplicate edges are allowed.
%        ED: (p,o) matrix, each row containing a (1,o) data vector associated
%           with the p'th edge.
%
%    g2: The second graph, identical format as g1.
%
%    K_vertex: A function handle realizing the vertex subkernel.  Takes two
%        vectors of dimension (1,m).
%
%    K_edge: A function handle realizing the edge subkernel.  Takes two
%        vectors of dimension (1,o).
%
% (The following parameters are optional and if they are ommited prob_S will
% produce a uniform distribution, prob_Q will be constant and prob_T will
% be uniform over all outgoing edges.  tol will be set to the default of 1e-6.)
%
%    prob_T: A function handle with parameters (g, to, from, fromQ), where g
%        is a graph, (from, to) are vertex indices specifying an existing edge
%        in the graph and fromQ is the probability of walk termination at the
%        vertex from.  prob_T returns the transition probability for the edge.
%
%    prob_Q: A function handle with parameters (g, i), where g is a graph and
%        i is a vertex index.  prob_Q returns the graph walk quit probability.
%        Alternatively, prob_Q can be a constant positive real c, 0 < c < 1.
%
%    prob_S: A function handle with parameters (g, i), g: graph, i: vertex
%        number.  prob_S shall return the random graph walk start probability
%        for the i'th vertex.  Can be ommited or set to [] for uniform
```

```
%        start probabilities.
%
%    method: 0 to use exact solution of linear system, > 0 to use the given
%        number as the number of iterations in the iterative scheme.  Default
%        is zero.
%
% Output:
%    K: The MGK kernel value.

g1len = size(g1.V, 1);
g2len = size(g2.V, 1);

% Produce r1
if isa(prob_Q,'function_handle')
    r1 = zeros(g1len*g2len, 1);
    for i=1:g1len
        for j=1:g2len
            r1((i-1)*g2len + j, 1) = prob_Q(g1,i) * prob_Q(g2,j);
        end
    end
elseif isa(prob_Q,'numeric') & length(prob_Q)==1
    % Constant termination probability
    r1 = ones(g1len*g2len, 1) .* prob_Q^2;
else
    error ('prob_Q','prob_Q must either be a function handle or a numeric.');
end

% Produce s, the (1,|g1|*|g2|) start probability vector.
s = zeros(1, g1len*g2len);
if nargin < 7 | length(prob_S) == 0
    % No start probability given, so we make it uniform
    for i=1:g1len
        for j=1:g2len
            s(1, (i-1)*g2len + j) = (1/(g1len*g2len)) * ...
                K_vertex(g1.V(i,:),g2.V(j,:));
        end
    end
else
    % Use start probability function.
    for i=1:g1len
        for j=1:g2len
            s(1, (i-1)*g2len + j) = prob_S(g1,i) * prob_S(g2,j) * ...
                K_vertex(g1.V(i,:),g2.V(j,:));
        end
    end
end
```

```
if nargin < 8
    method = 0;
end

% Produce the (|g1||g2|,|g1||g2|) coefficient matrix for the system of linear
% equations.

%T = zeros(g1len*g2len,g1len*g2len);
T = sparse(g1len*g2len,g1len*g2len);
edgecount1 = size(g1.E,1);
edgecount2 = size(g2.E,1);

% Because the matrix will be sparse, we explicitly use the arrays of edges to
% fill the matrix.
for f1index=1:edgecount1
    for f2index=1:edgecount2
        % We are certain that the t(t1,t2,f1,f2) term here exists and is
        % non-zero, so put it into the matrix.  First calculate it, then its
        % position.
        g1from = g1.E(f1index,1);
        g1to = g1.E(f1index,2);
        g2from = g2.E(f2index,1);
        g2to = g2.E(f2index,2);

        if isa(prob_Q,'function_handle')
            g1fromQ = prob_Q(g1,g1from);
            g2fromQ = prob_Q(g1,g1from);
        else
            g1fromQ = prob_Q;
            g2fromQ = prob_Q;
        end

        T_elem = prob_T(g1,g1to,g1from,g1fromQ) * ...
            prob_T(g2,g2to,g2from,g2fromQ) * ...
            K_vertex(g1.V(g1to,:), g2.V(g2to,:)) * ...
            K_edge(g1.ED(f1index,:), g2.ED(f2index,:));

        % Position: matrix is block-organised, every block has dimension
        % (|g2|,|g2|)
        col = (g1to-1)*g2len + (g2to-1) + 1;
        row = (g1from-1)*g2len + (g2from-1) + 1;
        T(row,col) = T_elem;
    end
end

% Iteratively, just to be sure, use a high number of iterations.
if method > 0
```

```
    R_inf = r1;
    for iter=1:method
        R_inf = r1 + T*R_inf;
    end
else
    % Solve: R_inf = R1 + T R_inf, for R_inf.
    R_inf = (speye(size(T)) - T) \ r1;
    %R_inf = gmres(speye(size(T)) - T, r1, [], 1e-4, [], [], [], r1);
end

K = s * R_inf;
```

# Bibliography


[1] K. P. Bennett and E. J. Bredensteiner, *Geometry in learning*, (1998).

[2] A. Bordes and L. Bottou, *The huller: A simple and efficient online SVM*, in ECML, 2005, pp. 505–512.

[3] S. Boughorbel, J. P. Tarel, and N. Boujemaa, *The intermediate matching kernel for image local features*, in IJCNN, 2005.

[4] S. Boughorbel, J. P. Tarel, and F. Fleuret, *Non-mercer kernels for SVM object recognition*, in British Machine Vision Conference, 2004.

[5] M. Brown and D. Lowe, *Invariant features from interest point groups*, in BMVC, 2002.

[6] C. J. C. Burges, *A tutorial on support vector machines for pattern recognition*, in Knowledge Discovery and Data Mining, vol. 2, 1998, pp. 121–167.

[7] K. R. Castleman, *Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, USA, 1996.

[8] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[9] O. Chapelle, P. Haffner, and V. Vapnik, *SVMs for histogram-based image classification*, Jan. 12 1999.

[10] S. Cohen, *Finding color and shape patterns in images*, PhD thesis CS-TR-99-1620, Stanford University, Department of Computer Science, May 1999.

[11] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, *Visual categorization with bags of keypoints*, Proc. ECCV International Workshop on Statistical Learning in Computer Vision, (2004).

[12] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, John Wily & Sons, Inc., second ed., 2001. ISBN 0-471-05669-3.

[13] J. Eichhorn and O. Chapelle, *Object categorization with SVM: kernels for local features*, tech. rep., Max-Planck-Institut für biologische Kybernetik, July 2004.

[14] J. D. Farquhar, S. Szedmak, H. Meng, and J. Shawe-Taylor, *Improving "bag-of-keypoints" image categorisation: Generative models and pdf-kernels*, tech. rep., LAVA, Feb. 17 2004.

[15] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, *Hypermedia Image Processing Reference (HIPR)*, John Wiley and Sons, 1997. ISBN 0-471-96243-0.

[16] T. Gärtner, *A survey of kernels for structured data*, SIGKDD Explorations, 5 (2003), pp. 49–58.

[17] P. Giannopoulos and R. C. Veltkamp, *A pseudo-metric for weighted point sets*, in ECCV (3), 2002, pp. 715–730.

[18] R. C. Gonzalez and R. E. Woods, *Digital image processing*, Prentice hall, Upper Saddle River, New Jersey, 2nd ed., 2001.

[19] M. Grabner, H. Grabner, and H. Bischof, *Fast approximated SIFT*, in ACCV (1), 2006, pp. 918–927.

[20] K. Grauman and T. Darrell, *Pyramid match kernels: Discriminative classification with sets of image features*, Tech. Rep. AIM-2005-007, MIT Artificial Intelligence Laboratory, Mar. 17 2005.

[21] B. Haasdonk and C. Bahlmann, *Learning with distance substitution kernels*, in DAGM-Symposium, 2004, pp. 220–227.

[22] C. Harris and M. Stephens, *A combined corner and edge detector*, in Proc 4th Alvey Vision Conf, Aug. 1988, pp. 189–192. Manchester.

[23] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, McGraw-Hill, 1995. ISBN 0-07-032018-7.

[24] T. Joachims, *Text categorization with support vector machines: learning with many relevant features*, in ECML, 1998, pp. 137–142.

[25] H. Kashima, K. Tsuda, and A. Inokuchi, *Marginalized kernels between labeled graphs*, in ICML, 2003, pp. 321–328.

[26] Y. Ke and R. Sukthankar, *PCA-SIFT: A more distinctive representation for local image descriptors*, in IEEE Computer Vision and Pattern Recognition or CVPR, 2004, pp. II: 506–513.

[27] R. I. Kondor and T. Jebara, *A kernel between sets of vectors*, in ICML, 2003, pp. 361–368.

[28] T. Lindeberg, *Scale-space theory: A basic tool for analysing structures at different scales*, Journal of Applied Statistics, 21 (1994), pp. 224–270.

[29] ——, *Feature detection with automatic scale selection*, International Journal of Computer Vision, 30 (1998), pp. 79–116.

[30] T. Lindeberg and J. Gårding, *Shape-adapted smoothing in estimation of 3-D depth cues from affine distortions of local 2-D brightness structure*, in ECCV (1), 1994, pp. 389–400.

[31] ——, *Shape-adapted smoothing in estimation of 3-D shape cues from affine deformations of local 2-D brightness structure*, Image and Vision Computing, 15 (1997), pp. 415–434.

[32] D. G. Lowe, *Object recognition from local scale-invariant features*, in International Conference on Computer Vision, 1999, pp. 1150–1157.

[33] ——, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision, 60 (2004), pp. 91–110.

[34] S. Lyu, *Mercer kernels for object recognition with local features*, Tech. Rep. TR2004-520, Dartmouth College, Computer Science, Hanover, NH, Oct. 2004.

[35] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, *Extensions of marginalized graph kernels*, in ICML, 2004.

[36] K. Mikolajczyk, B. Leibe, and B. Schiele, *Local features for object class recognition*, in ICCV, 2005, pp. 1792–1799.

[37] K. Mikolajczyk and C. Schmid, *Scale & affine invariant interest point detectors*, International Journal of Computer Vision, 60 (2004), pp. 63–86.

[38] ——, *A performance evaluation of local descriptors*, IEEE Trans. Pattern Analysis and Machine Intelligence, 27 (2005), pp. 1615–1630.

[39] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. van Gool, *A comparison of affine region detectors*, International Journal of Computer Vision, 65 (2005), pp. 43–72.

[40] P. J. Moreno, P. Ho, and N. Vasconcelos, *A kullback-leibler divergence based kernel for SVM classification in multimedia applications*, in NIPS, 2003.

[41] P. Pedregal, *Introduction to Optimization*, Springer-Verlag, New York., 2004. ISBN 0-387-40398-1.

[42] J. Ramon and T. Gärtner, *Expressivity versus efficiency of grapgh kernels*, in Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences (MGTS-2003), L. De Raedt and T. Washio, eds., ECML/PKDD'03 workshop proceedings, Sept. 2003, pp. 65–74.

[43] Y. Sawaragi, H. Nakayama, and T. Tanino, *Theory of Multiobjective Optimization*, Academic Press, Inc., 1985. ISBN 0-126-20370-9.

[44] C. Schmid and R. Mohr, *Local grayvalue invariants for image retrieval*, IEEE Trans. Pattern Anal. Mach. Intell, 19 (1997), pp. 530–535.

[45] B. Schölkopf, A. J. Smola, and K.-R. Müller, *Nonlinear component analysis as a kernel eigenvalue problem*, Neural Computation, 10 (1998), pp. 1299–1319.

[46] B. Schölkopf and A. J. Smola, *Learning with Kernels, 2nd Edition*, MIT Press, 2002. ISBN 0-262-19475-9.

[47] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004. ISBN 0-521-81397-2.

[48] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman, *Discovering objects and their localization in images*, in ICCV, IEEE Computer Society, 2005, pp. 370–377.

[49] M. Šonka, V. Hlaváč, and R. D. Boyle, *Image Processing, Analysis and Machine Vision*, PWS, Boston, USA, second ed., 1998. ISBN 0-534-95393-X.

[50] A. Stein and M. Hebert, *Incorporating background invariance into feature-based object recognition*, in Workshop on Applications of Computer Vision, 2005, pp. I: 37–44.

[51] K. Tsuda, *Support vector classifier with asymmetric kernel functions*, Oct. 18 1999.

[52] K. Tsuda, T. Kin, and K. Asai, *Marginalized kernels for biological sequences*, in ISMB, 2002, pp. 268–275.

[53] N. V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York., 2000. ISBN 0-387-98780-0.

[54] C. Wallraven, B. Caputo, and A. Graf, *Recognition with local features: the kernel recipe*, in International Conference on Computer Vision, 2003, pp. 257–264.

[55] J. Weston, B. Schölkopf, E. Eskin, C. Leslie, and W. S. Noble, *Dealing with large diagonals in kernel matrices*, 2002.

[56] J. Weston and C. Watkins, *Multi-class support vector machines*, 1998.

[57] A. P. Witkin, *Scale-space filtering*, in Proceedings of the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, FRG, Aug. 1983, William Kaufmann, pp. 1019–1022.

[58] L. Wolf and A. Shashua, *Learning over sets using kernel principal angles*, Journal of Machine Learning Research, 4 (2003), pp. 913–931.

# Index