Master's Thesis

Sparse Coding for Video-Based Analysis

Max Stricker

Institute for Computer Graphics and Vision
Graz University of Technology
A-8010 Graz

Assessor: Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof
Advisor: Dipl.-Ing. Dr.techn. Peter Roth

Graz, April 2012

Acknowledgements

This master's thesis has been conducted in 2012 at the Institute for Computer Graphics and Vision, Graz University of Technology.

First of all I would like to thank everyone who supported me during the elaboration of my master's thesis: The institute for Computer Graphics and Vision and Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof provided me the possibility to participate in ongoing research in the institute in the area of computer vision, and to join the pleasant working atmosphere.

During the development of my thesis I was supported by Dipl.-Ing. Dr.techn. Peter Roth, who issued various ideas and assisted me in finding solution strategies to solve upcoming problems. Writing a paper and attending the Computer Vision Winter Workshop 2012 conference has been a great experience.

I would also like to thank everyone else who supported me during the workout of my thesis, which I did not mention here by name.

This thesis was supported by the Austrian Research Promotion Agency (FFG) under the FIT-IT project OUTLIER (820923).

Graz, April 2012

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Abstract

An important step for each object recognition task is to find an adequate representation of the information contained in an image. This representation can be generated through analyzing the object's appearance or geometry, and depending on the task needs to fulfill certain requirements. Recently, research has evidenced that sparse feature descriptors perform well for many object recognition applications. Therefore Sparse Coding, a sparse feature descriptor representing images as sparse linear combinations of a few basis elements, has been investigated in this work in order to emphasize its strengths and drawbacks for different computer vision tasks. The first goal has been to theoretically analyze Sparse Coding and its main parameters. Various experiments have been conducted to reason about different parameter settings and their implications. The evaluation of these experiments made it possible to derive the importance of various parameters for the feature descriptor's expressiveness. The second part of this master's thesis addresses the applicability of Sparse Coding for the analysis of video sequences based on two exemplary application scenarios. The first application deals with the problem of needing many labeled images when training classifiers for object recognition. This work proposes a general method, based on Sparse Coding, to automatically label individual frames within natural video sequences. These labels are generated by analyzing the slow driving force within the sparse feature descriptor. Experiments on two datasets have been performed to evaluate the proposed system. Based on the results of the analysis of Sparse Coding, a framework for detecting unusual events and irregularities in videos has been developed. This unsupervised task has been chosen to identify and assess the strengths and weaknesses of Sparse Coding compared to other state-of-the-art feature descriptors. This work emphasized benefits when using Sparse Coding for different computer vision tasks dealing with video-sequences.

Keywords: Object recognition; Sparse Coding; Video analysis; Automatic labeling; Unusual event detection;

Kurzfassung

Ein wichtiger Schritt für jedes Objekterkennungssystem ist das Finden einer geeigneten Beschreibung für die in den Bildern enthaltene Information. Diese Repräsentation kann durch die Analyse der Erscheinung oder der Geometrie eines Objektes berechnet werden, und sollte je nach Verwendungszweck verschiedene Anforderungen erfüllen. Aktuelle Publikationen zeigen, dass Sparse Beschreibungen für Objekterkennungssysteme sehr gut geeignet sind. Deshalb wird im Rahmen dieser Masterarbeit Sparse Coding, ein spezielle sparse Beschreibung, welche das Bild als dünnbesetzte Linearkombination von Basiselementen darstellt, behandelt, um dessen Eigenschaften für verschiedene Bilderkennungssysteme zu analysieren. Der erste Teil dieser Arbeit umfasst daher die theoretische Analyse von Sparse Coding und seiner wichtigsten Parameter. Verschiedene Experimente und Analysen wurden durchgeführt, um Aussagen über die Parameter und ihre Eigenschaften zu treffen. Der zweite Teil dieser Arbeit widmet sich der Umsetzung zweier Objekterkennungssysteme basierend auf Videosequenzen. Im ersten Anwendungsfall wird auf das Problem eingegangen, dass für das Trainieren von Klassifikatoren zur Objekterkennung vielegekennzeichnete Bilder benötigt werden. Dieser Arbeit stellt eine generelle Methode, basierend auf Sparse Coding, zur automatischen Generierung solcher Kennzeichnungen aus Videos vor. Diese Kennzeichnungen werden durch die Analyse der sich über die Zeit verändernden Beschreibungen berechnet und durch Experimente auf zwei verschiedenen Datensätzen validiert. Basierend auf den Analyseergebnisse des Sparse Coding, wurde als zweite Applikation ein System zur Erkennung von Unregelmäßigkeiten in Videosequenzen entwickelt. In diesem System wurden die Stärken und Schwächen von Sparse Coding im Vergleich zu anderen, dem Stand der Technik entsprechenden Bildbeschriebungsmethoden analysiert. Zusammenfassend zeigt diese Arbeit Stärken von Sparse Coding für verschiedene videobasierte Anwendungsfälle in der Bilderkennung.

Stichwörter: Objekterkennung; Sparse Kodierung; Videoanalyse; Automatisches Labeling; Erkennung unüblicher Ereignisse;

Contents

1 Introduction

1.1 The Role of Computer Vision

> "A picture is worth a thousand words"
>
> Fred R. Barnard, 1921

In today's technology-enhanced society, a vast amount of information in various forms, such as text, videos and images, is generated and consumed every day. In order to deal with this high quantity of information, it is desirable to use automatic tools and methods to support human consumers.

Within the area of text processing or mining, technologies, such as information extraction or search, automated summary generation, machine translation and others, are used in various application domains. Coexisting, an analogous research field deals with information present in images and its understanding, called Computer Vision.

Computer Vision generally describes the research field developing theories and applications with the primary goal to automatically analyze information present in images. A variety of fields, including, but not limited to mathematics, signal processing, geometry pattern recognition and artificial intelligence, interact together with the aim to create better tools for describing and interpreting images.

The interpretation of the embedded information within those images allows - depending on the use case and task - to support or automate processes in different application areas.

In the area of medical and health care, images coming from a variety of medical equipment (computer tomography, x-ray, ultrasonic, microscopy, ...) play an essential role in prevention, diagnosis and treatment of diseases and their progress. Various computer vision systems [90, 29, 60, 24] support human experts in decision making. Such systems are able measure important parameters about internal organs (like bone structure, tumor size, ...) without the need of surgical interventions.

Many military operations are associated with a high risk for humans. Thus, computer vision is often used to control robots which can be used to oversee combat actions, to explore enemy territory, to identify hazardous situations or to automatically navigate missiles to their

targets [97]. Similar risky actions can also be supported on space missions [67] and other environments hostile to man.

Industrial applications contain also rich usage possibilities for computer vision supported technologies. Well-known examples include automatic parcel sorting according to its size, estimated from a camera image, or other component inspection tasks. Computer vision systems are widely adopted for automated quality control, where photos taken from parts passing a control point on a production line are used to check for specified quality criteria.

The automotive sector has recently begun to incorporate various computer vision techniques to support drivers and increase security. Cameras can detect street signs, calling the driver's attention in case of possible traffic violations. To decrease the chance of traffic accidents or crashes, cameras inspect the car surroundings for pedestrians, cars or other objects. If such a monitored object could cause a dangerous situation, the system intervenes by triggering an emergency brake.

Alongside with the adoption in industry applications, computer vision has recently become very popular in consumer products. Many modern, consumer-grade digital cameras provide different features to detect faces, focussing on them and automatically taking a picture when the person smiles. Photo organization software like Apple's iPhoto [1] can automatically detect persons and localize faces in photographs. Photo recognition is used to automatically group photos containing the same person, learned from a few user-provided samples.

Also smartphones with their variety of sensors and high computational power are well suited for computer vision applications. Augmented reality applications play an important role on these devices where the goal is to enrich the perception of the real world with additional information. Based on the information used for enrichment, these systems can serve for various purposes, from entertainment to navigational support, or even to assist during medical surgeries.

Different security-related duties can also be simplified or enhanced by computer vision. Human motion analysis [2] in security-sensitive areas can be used to assist security service personnel to identify potential threats.

Many of the presented usage scenarios are fairly easy for humans. They are efficient in identifying and recognizing objects from few samples. Infants are able to understand the concept of, e.g., a car or a dog, even after having seen only few samples. For most people it is also an easy task to re-identify a person after having seen just one picture of her. Even appearance changes due to different hair styles or a recently grown mustache don't lead to problems in this process.

Surprisingly the same tasks (person identification from one sample) is still a hard-to-solve problem for computer vision algorithms. This stresses the need for ongoing research in computer vision.

[1] `http://www.apple.com/ilife/iphoto/`

Until now different application scenarios for vision-centric algorithms have been motivated without any further details about the involved processes. As in [5] a typical computer vision system can be represented as a multistage process involving the following steps:

- **Sensing** describes the process that leads to the actual image(s), which can come from a multitude of sources (camera systems, medical equipment) by capturing the current environmental conditions.
- **Preprocessing** includes enhancement steps on the previously sensed images. This task consists of a series of operations, like noise reduction, distortion removal, resolution change, and other quality enhancements.
- **Segmentation** deals with the separation of an image into different segments or regions or interest. The most common regions or interest are foreground and background information.
- **Description** - as the name already proposes - is the task in which the image is described by its characteristics. Details of these feature descriptions are mentioned in chapter 2.
- **Recognition** treats the process of identifying and localizing objects of interest within the described images.

The combination of the last of these two tasks - namely description and recognition - is usually referred to as object recognition. Describing the content of an image is one of the most crucial parts: without a good description of the image, it is extremely hard to recognize objects in this image in the later recognition stage. Nevertheless, object recognition includes also aspects from the other tasks (sensing, preprocessing and segmentation), as these ones influence the results of image description and recognition.

Object recognition is commonly understood as the task of finding and localizing objects within an image. In order to effectively perform this task, prior knowledge about the object of interest is needed [92]. This knowledge has to be modeled prior to the actual task of recognition.

Depending on the task, an object recognition system has to fulfill certain requirements. One crucial aspect for an object recognition system is the execution and evaluation time it takes to detect all objects within an image. When analyzing a production line for faulty produced parts, the time needed for inspecting is crucial in order to maximize the number of produced and analyzed parts per day. On the other hand, recognition reliability plays also an important role to minimize false positives (correct parts detected as faulty) and the false negatives (faulty parts detected as working). Another quality measure for those systems is the system's invariance to changes. It is desirable that the system is invariant to different variations of the object to be detected, like affine transformations, illumination changes or occlusions.

Due to the different requirements, constraints and application scenarios for object recognition, there exist a variety of approaches, that don't fit into an exact categorization, but can be analyzed and partially classified according to the following properties [92]:

One property, decided on during the description stage, is the object representation, where mainly two different approaches exist. The first class of descriptors are based on the object's geometry and therefore describe it based on its surface, shape or silhouette. In contrast, appearance-based image descriptors interpret the object depending on the object region characteristics.

In order to precisely detect and locate a modeled object within the described image, a robust matching strategy is needed. This matching strategy compares the current image with the modeled objects and - by maximizing a similarity measure, computed on the correspondence between model and scene - decides whether the scene contains the objects.

Another important aspect considers the expected object variability to be recognized and worked with. On a product line, objects belonging to the same class are awaited to share many characteristics, therefore having a low class-internal variance. In other applications, objects belonging to the same class could exhibit various deformations, leading to a high class internal variance. On the other hand, when trying to distinguish multiple object classes, the variance between different categories is crucial.

Object recognition, i.e. image description and recognition, is also the main topic covered in this master's thesis, and will be motivated in the following section.

1.2 Motivation and Aim of this Work

Many technological devices are equipped with video capturing capabilities and it is easily and cost-effectively possible to create new video material. Such devices include smartphones, tablet pcs and laptops. Video surveillance systems are also installed in modern cars, in stores, companies and public places in order to monitor activities and behaviour, and create a waste amount of video material every day.

Controlling this huge amount of information entirely by humans would become infeasible. Therefore computer vision algorithms try to reduce or eliminate the manual work and intervention needed to operate such systems.

To operate efficiently on those video sequences, it is crucial to build a good and robust description of its content. In recent publications it has been shown that sparse feature descriptors lead to promising results.

Therefore, the main aim and goal of this master's thesis was to analyze and apply Sparse Coding, a sparse feature descriptor, on video sequences. Sparse Coding has been analyzed

through various experiments, resulting in a set of guidelines to follow when using Sparse Coding. They highlight the importance of spatial pooling, the process of combining many low-level descriptors into one, and the used codebook's size and quality. In addition to theoretically derive and analyze sparse coding, its application for two different video-based application scenarios has been tested and evaluated.

The objective of the first application was to introduce and analyze the possibility to automatically generate class labels for individual video sequence frames, based on the temporal change over time. This allows for fast and efficient automatic label generation, useful when training object classifiers. In this part, a novel Sparse-Coding based approach for image labeling using Slow Feature Analysis has been proposed, that, based on this robust description, is able to cope with large appearance changes.

In the second task of this work, a simple yet effective methodology for detecting unusual events in unlabeled video streams is introduced and evaluated.

In order to be able to better highlight the characteristics, i.e., the advantages and disadvantages, of sparse coding, the two tasks have not only been performed using sparse coding, but also other well-known feature descriptors have been used. In this way, the performance of the different descriptors can be compared directly. The results indicate, that Sparse Coding provides a robust mid-level representation, providing better results than other appearance-based feature descriptors.

1.3 Thesis Outline

To analyze and validate the usage of Sparse Coding as compact, robust mid-level feature representation, this master's thesis is composed of several parts. First of all, popular low-level features and their relation to Sparse Coding are introduced and shortly analyzed. The main part of this work then defines Sparse Coding formally to highlight the required information needed to understand the concepts behind it. The whole coding pipeline, the process to derive a sparse mid-level representation from local low-level features, including the visual codebook learning and spatial pooling is discussed and validated through various experiments.

Based on this theoretical knowledge, the power of Sparse Coding is demonstrated on two different, video-related tasks. One shows a novel approach to automated video labeling based on Slow Feature Analysis and the robust Sparse Coding descriptor, by analyzing the temporal feature change. The second application uses Sparse Coding as robust feature descriptor to describe natural video scenes, from which abnormalities and unusual events are detected. At the end, the main goals and results are summarized, and future work is stimulated.

In terms of chapters, the remainder of this thesis is structured as follows:

2 Image Descriptors: This chapter motivates the need for feature descriptors in computer vision and its main desirable properties. Exemplary a few important feature descriptors that are used in later experiments are summarized and compared.

3 Sparse Coding: In this chapter Sparse Coding is derived from the general notion of matrix factorization, its coding-pipeline including state-of-the-art algorithms are introduced. It also contains various experiments and results on the most important parameters of this method like codebook size and quality or the choice of pooling strategies.

4 Applications: Two applications demonstrate the applicability of Sparse Coding to video sequences. In the first application an unsupervised method to automatically detect object class changes and label the individual frames is introduced and different experimental studies are performed. The second application's aim is to detect abnormal events in video streams and report them. A system based on Sparse Coding is introduced and comparative studies to other feature descriports are evaluated.

5 Conclusion: This chapter summarizes the main findings and provides an outlook for possible future work in this area.

2 Image Descriptors

2.1 Introduction

Feature descriptors and their change over time form the basis of this masters thesis, therefore this chapter introduces the usage of feature descriptors for computer vision applications.

First, general properties of feature descriptors and their importance are discussed and motivated. Thereafter follows a short survey on state of the art feature descriptors, their strengths and weaknesses. Exemplary for other feature descriptors the Scale Invariant Feature Transform, the Histogram of Oriented Gradients, the Histogram of Oriented Optical Flow and the bag-of-words model for computer vision, are introduced as further experiments build on these descriptors.

A (local) feature descriptor or feature vector is a piece of information different from the one available in the immediate image neighborhood[94].

This definition already opens a few important questions. The first important aspect is how the piece of information is represented. As the second term "*feature vector*" indicates, that this information is usually stored as a vector of single signals or values. Other issues are how to measure the difference to the neighborhood and how to define the location of the feature within the image.

Depending on its usage, a feature descriptor needs to fulfill different requirements. For some use cases these feature descriptors might encode specific semantic information: edges within aerial images might indicate roads, or edges might be useful when detecting production faults in an assembly line. For another use case it is not even relevant what information a feature encodes. For many tracking and matching applications it is just of importance to identify and precisely locate individual feature points. It is more important to locate the feature descriptor over time than to know the information it encodes. A set of such features can also be used as an image representation, useful for object recognition tasks. Again it is not needed to know the exact meaning of a single feature, as an image is described based on the statistics of those features.

Among others, feature descriptors can be analyzed and categorized based on the following characteristics:

- Invariance
  It is desirable that a feature descriptor is invariant to specific changes in the input image, meaning that the feature descriptor for the modified image should be close to the one for the original image. These modifications include changes in scale, lightning, rotations, affine transformations or combinations of these.
- Efficiency
  Preferably the detection and computation of features from an image should be efficient and suitable for time-critical or realtime applications.
- Distinctiveness
  Two feature descriptors describing distinctive information should easily be distinguished based on their feature values.
- Quantity
  The number of extracted features from an image should be sufficiently large to represent even subtleties in small image regions, but a high number of features normally has a negative impact on computational complexity. Ideally the quantity of local image features should be correlated to the amount of information present in the image.

Clearly these requirements and characteristics are problem dependent; when having a problem without time constraints it might be advisable to trade efficiency for more invariance.

As there are quiet a number of different feature descriptors, it would not be feasible to explain them all as part of this master's thesis. Thus, the ones used later on in various experiments, have been chosen to be discussed in more detail in the next sections.

2.2 The Bag-of-Words Model

As already motivated, this thesis analyzes the mid-level feature representation Sparse Coding. It is called mid-level, because it uses a set of low-level image descriptions and combines them in a clever way to form a single description for that image. Sparse Coding is, however not the only form of mid-level feature representation. Without any particular specification of how a low-level feature looks like or how it is computed, another mid-level feature representation called the Bag of Words model is introduced, as it gives a good intuition about how one can combine single descriptors to form a more high-level representation. The later sections then describe basic, low-level features that can be used with a Bag of Words model, and are used throughout this work.

The Bag of Words (BoW) model, often also called Bag of Features model, describes a popular method for data representation originating within the field of natural language processing [85]. In natural language processing, text documents are often represented by their keywords, regardless of the order they appear. After some optional initial preprocessing, like stop word removal or stemming, a dictionary is created using all keywords from all documents.

Afterwards each document is represented by the count of the individual words within the dictionary.

The following simple example illustrates these steps and the final feature representations.

Given the following sentences:

- **Sentence 1:** "John likes computer vision research. So does Mary."
- **Sentence 2:** "John also likes writing papers on computer vision."

From these sentences the following dictionary is constructed:

- {"john","likes","computer","vision","research","also","writing","papers","on", "so","does","mary"}

Based on this codebook each sentence is then represented by the count of the codebook entries:

- **Sentence 1:**{1,1,1,1,1,0,0,0,0,1,1,1}
- **Sentence 2:**{1,1,1,1,0,1,1,1,1,0,0,0}

The name bag of words comes from the fact that each sentence is represented by a bag of words, loosing the original order.

The same concept is also popular as image representation in computer vision applications. The concept of a document is replaced by an image, and words are represented by some features.

In [62] Lowe presented a revolutionary method to describe local image regions and how to effectively match them. For many applications however, a single, more high-level image representation might be desirable. By representing an image as orderless set (bag) of local features, in form of a histogram, the bag-of-words-model provides a simple and effective mid-level feature representation [50, 73].

The process to create a bag-of-words model for a computer vision application is therefore slightly different, still following the same principles.

First local features are extracted from the image, on predefined locations. These locations could be chosen randomly, but better results can be achieved using regular grids or some sort of interest points (corners, edges,...). The bag-of-words model does not specify how these local features should be described, a popular choice is SIFT, but any other feature descriptor is also usable. At the end of this process each image is represented by a collection of feature vectors.

Based on these features a codebook is constructed, where a simple methodology clusters the feature descriptors. The codebook is then composed of the cluster centers, meaning that one codebook entry is a representative description for many local features. When creating the final feature descriptor, each local feature is mapped to a codebook entry (the nearest cluster center), thus creating the final histogram representation. Normally this histogram feature is then normalized, to eliminate the influence of the number of local features extracted, as they might be different when using interest point detectors.

One of the drawbacks of this approach however is, that the spatial information where a specific codebook entry appeared in the image is lost. Also the relation between codewords in the source image can not be expressed with this simple model. To overcome this problems several publications [82, 87] propose different extensions to incorporate such information.

2.3 Scale Invariant Feature Transform

As mentioned previously, there exists a large number of feature descriptors, of which one, Scale Invariant Feature Transform (SIFT), is presented in the following. SIFT has been chosen because many of the topics discussed later on in this work are built upon this popular descriptor, and it is therefore quiet important for the understanding of this work to know these concepts.

Scale Invariant Feature Transform, or simply SIFT, is a feature descriptor which was defined by David Lowe in 1999 [61]. It is a very popular feature descriptor, used for various tasks such as object recognition, video tracking, panorama stitching, or gesture recognition.

Based on [88] the process of computing a SIFT descriptor for a single location is described.

First of all, the interest points (or key points) have to be selected. This can be done either randomly, or by using one of many existing techniques. For example, these key points can be extracted on a fixed-sized grid. There also exist special purpose algorithms for key point localization, which try to estimate the most interesting image parts to describe the current scene. Such relevant locations could include corners, edges or other properties that differ from the surroundings.

For each interest point inside the image, a 16×16 pixel window around this point is used for the computation of the descriptor, as shown in figure 2.1. For each pixel within this window, the gradient magnitudes and orientations are computed. They are weighted using a circular fall-off function (e.g., a Gaussian) in such a way, that values located far away from the interest point have lower influence than the nearer ones.

The mentioned window is then partitioned in 16 disjoint subregions of size 4×4. After that, each of these subregions is processed by computing a gradient oriented histogram, where the

degrees are binned and each of the previously computed gradient vectors is added to a bin, respectively to its orientation. This is done by using its magnitude, weighted by the distance fall-off function. As a result, one gets a total of 128 non-negative values representing the "raw" Scale Invariant Feature Transform descriptor.

For contrast invariance in the description of the input image, the descriptor is then normalized (to length 1). To be also robust to other photometric variations, the descriptors' individual values are cut off at 0.2, meaning that all values greater than this threshold are diminished to 0.2, and after this clipping the descriptor is again normalized to unit length.

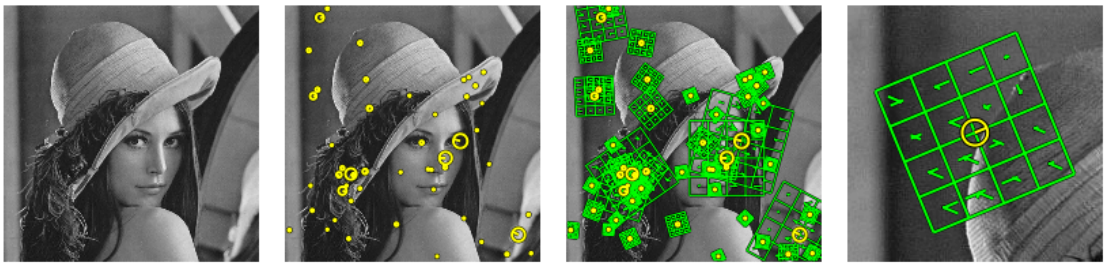

Figure 2.1: Illustration of a SIFT feature descriptor containing the input image, detected key points, the key points including the 16x16 pixel window and one such window and its subregions.

Due to its popularity, SIFT is widely used in various applications. However, its relatively high runtime (due to the computation of the gradient) motivated many scientists in this field to develop various extensions, like PCA-SIFT and SURF. PCA-SIFT [43] computes all gradients within a 39×39 pixel window and then reduces the vector dimension using principal component analysis. SURF [4] speeds up the SIFT computation by eliminating the Gaussian filters and uses faster-to-compute box filters with integral images. Grabner et al [34] explain another fast approximation of SIFT, also relying on integral images.

2.4 Histogram of Oriented Gradients

This section describes how a histogram of oriented gradients also called HOG, is generated and why it can be used for action and object recognition in practice.

Histograms of oriented gradients were originally developed by Dalal and Triggs in 2005 to have a feature descriptor for human poses [23]. In their work each HOG describes an individual human pose, or more generally spoken, the current appearance or state visible in an image. Therefore, successive poses can be described by a series of HOGs. In this way, and as they are quiet robust to various changes, HOGs can be successfully applied to various tasks dealing with object [13], and action recognition [81, 59].

Dalal et al [23] state that the content of an image can usually be described by only using the distribution, omitting the precise locations, of the gradients (or edge directions) inside the image. According to this, an overview over the process of computing the HOG descriptor of an input image is shown in figure 2.2.

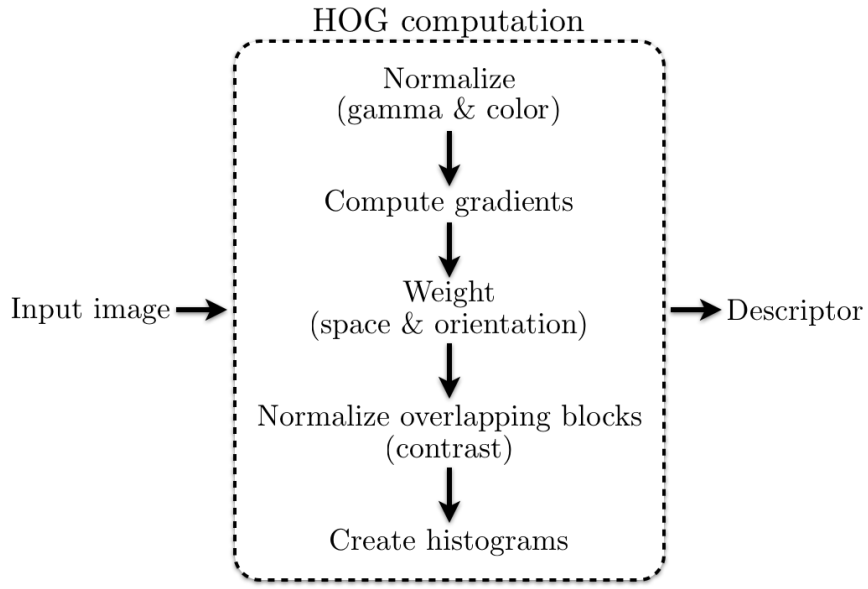

Figure 2.2: Overview over the process of computing a histogram of oriented gradients descriptor. Figure adapted from Dalal et al, originally presented in [23].

First of all, the input image window (of size 64×128 pixels in the original implementation) is tiled into partially overlapping larger spatial regions, called blocks, of size 16×16 pixels, each of which contains four 8×8 pixel cells. This overlapping of the blocks, for which the HOGs will be computed, leads to a dense covering of the input image. The main focus during the computation of a histogram of oriented gradient lies, as the name already proposes, on the computation of the gradients. For each pixel inside a cell, the gradient directions or edge orientations are computed, forming a local histogram. It should be noted, that when using color images, the gradients of all channels have to be computed individually, and the gradient vector with the highest norm is then used to describe that pixel. In the original implementation of Dalal et al, 9 orientation bins evenly cover the range from 0 to 180° (unsigned gradient orientations). It is however also possible to include the entire span from 0 to 360°, which represents the signed orientation of the gradient. By accumulating the local histograms of one block, an "energy" value can be obtained, which is used to normalize according to gamma and color values, such that invariance of the descriptor to changes in the illumination can be ensured. Based on the gradient orientation and position, for each pixel a weighted value is computed, and some interpolation is performed between neighboring pixels to have anti-aliasing. Putting these values together for all pixels inside a cell, results in an orientation histogram. The overlaps of blocks ensure, that the different cells influence

various neighboring blocks. In order to make the final descriptor contrast invariant, these overlapping blocks are additionally normalized.

Such a descriptor in form of a histogram can thus be used to describe single states or objects, or, when used in a sequence, actions going on in sequential images (or videos).

2.5 Histogram of Oriented Optical Flow

In this section, the concept of optical flow and its extraction from video sequences will be discussed. Furthermore, it is described how different extensions can be applied to the extracted optical flow feature, in order to make it more robust and invariant, resulting in a so-called histogram of oriented optical flow.

Optical Flow (OF)

The most intuitive and natural feature to characterize video sequences is the optical flow, which describes the motion in a sequence of images (or video frames) [17]. Video motion cannot be captured and represented by low-dimensional models (such as parametric motion models), which is why the optical flow, a directional feature with an associated magnitude value (similar to the gradient), has to be computed [88, 18].

Generally, optical flow is understood as computing an independent estimate of motion at each single pixel of the video frames. Theoretically, this can be done, as shown in figure 2.3, by using the image values at a certain time t and $t + \delta$, and by computing the partial derivatives with respect to space and time coordinates at each image position, i.e., at each pixel. The motion value is then obtained by summing the minimal brightness or contrast values over the entire image (or video frame). However, this minimization problem is under-constrained, which is why various approaches, like described in [88], are used to estimate the true values.

Even though the optical flow seems to be an appropriate feature, it still lacks some quiet important properties [17]: A first problem is that the number of pixels describing the object of interest, and thus the descriptor size, changes over time. Additionally, various factors, like the background noise, changes in the scale of the moving objects (by moving towards or away from the camera, or by zooming in and out), and the direction of the motion (i.e., moving from left to the right or vice-versa), influence the resulting features.

In order to overcome some of these drawbacks, one might think of using a distribution of optical flow vectors. But still, the problem of missing the invariance of motion direction and scale changes remains the same.

The desired (optimal) motion feature should be based on optical flow, such that the motion at all points in time are catched, but it should also be more robust to changes in the scale

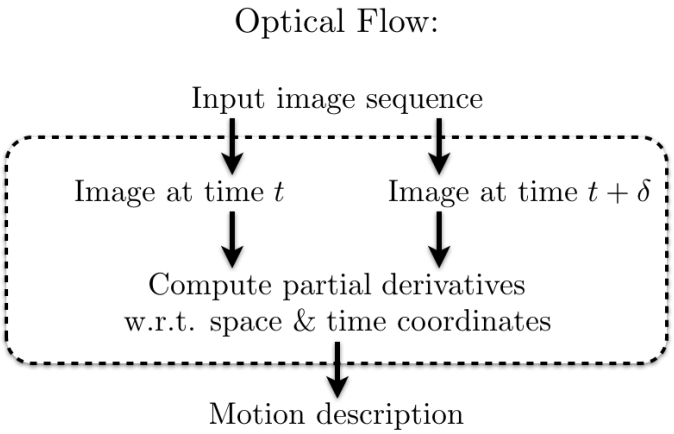

Figure 2.3: Overview over the extraction of motion information in form of optical flow of a sequence of input images.

and to the direction of the movement. Thus, the expected descriptor should be quiet similar to the histogram of oriented gradients described in the previous section 2.4. This is where one can refer to histograms of oriented optical flow, which are described in the following.

Histogram of Oriented Optical Flow (HOOF)

The so-called histogram of oriented optical flow, or simply HOOF, is a feature descriptor which fulfills the requirements stated previously [17]: It is able to characterize human motion, or, when using a time series of HOOFs, to represent movements in image sequences or videos.

The methodology how to come up with such a histogram is the same as for HOGs (described in section 2.4), with the only difference that for HOOFs the optical flow has to be extracted from the image sequences, and not only the gradients. This process is briefly described in the following, and also shown in figure 2.4.

The first step in the construction of a HOOF is intuitively to compute the optical flow vectors for each frame in a video sequence.

In a second step, the computed OF vectors are taken and binned according to their primary angle. This primary angle describes the minimum of the unsigned values of the angles from the optical flow vectors to the horizontal axis. This step, i.e., taking the unsigned angles, makes the descriptor invariant to the direction of the motion in the video, meaning that the descriptor is the same for the situation where a person runs from the left to the right, and where the person runs from the right to the left. The number of bins for the angle degrees of the optical flow vectors is a parameter one can choose freely; but according to [17] numbers above 30 lead to good results.

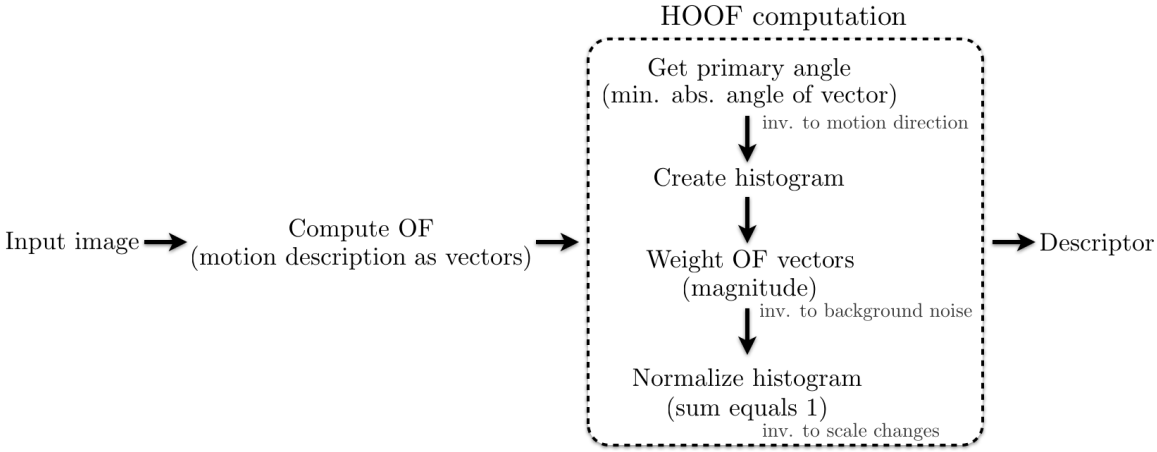

Figure 2.4: Overview over the process of computing a histogram of oriented optical flow descriptor, by first computing the optical flow, and then creating a histogram as the final descriptor out of the flow vectors.

Additionally, each flow vector is weighted according to its magnitude. This avoids (or at least reduces) the influence of the background of the scene significantly: Background noise up to a certain level is ignored, as its flow magnitude is much lower than the one of the moving object in the foreground.

In a last step, the obtained histogram is normalized to sum up to one, which finally makes the descriptor invariant to scale changes of the object in motion.

3 Sparse Coding

This chapter introduces the concept of Sparse Coding by first giving a broad overview and insight on its recent use in computer vision algorithms in chapter 3.1. The following chapter 3.2 , after a short introduction to numerical optimization, defines sparse coding as a convex optimization problem and gives some insight on how to solve it in practice. Chapter 3.3 describes the whole process of constructing a sparse descriptor for an image, including codebook learning and spatial pooling. The last chapter contains experiments and results, conducted to analyze the main parameters of Sparse Coding.

3.1 Definition

Put into one sentence, sparse coding describes a theoretical framework for modeling data vectors as linear combinations of few elements from a basis set often called dictionary.

Sparse coding is not a particular method used for learning feature descriptors but a general methodology to generate sparse signals. In fact it has been first published in 1995 in [27] within the area of biology.

One can interpret a brain as a special form of computing machinery where information is processed. It receives external sensory input, like for instance from the visual system and transforms it into an appropriate motion output. Since it is obvious that in such a computing system information has to be processed, the brain must be able to store an internal representation of the mentioned information. In a brain such information is represented and transmitted as electrical firing patterns of single neurons. It must therefore be possible for the brain to express and encode all internal and external environmental states using the firing patterns of a fixed set of neurons. The number of neurons can possibly be large, like for example the human brain contains about 100 billion such connected neurons [38], but its amount is fixed. If one for simplicity assumes that each neuron can only be in a "firing" or "not firing" state and the neurons have a low activity ratio, meaning that only a small fraction of all neurons is firing at any given time step, this firing pattern forms a sparse code.

The concept of expressing input data as linear combination of basis elements is illustrated in figure 3.1. The 16×16 pixel image patch on the top is extracted from the photograph on the left and the goal is now to express this image patch by a few other image patches. The box on the bottom is the basis set (dictionary, codebook, vocabulary) which is used to approximate

the input patch. One possible approximation is to take 0.41 times the pixel intensities of codebook patch 9, 0.11 times codebook patch 27 and 0.16 times codebook patch 50.

If we put the contribution of each codebook element into a vector only the elements at position 9,27 and 50 are different from 0 and this code is therefore called a sparse vector.

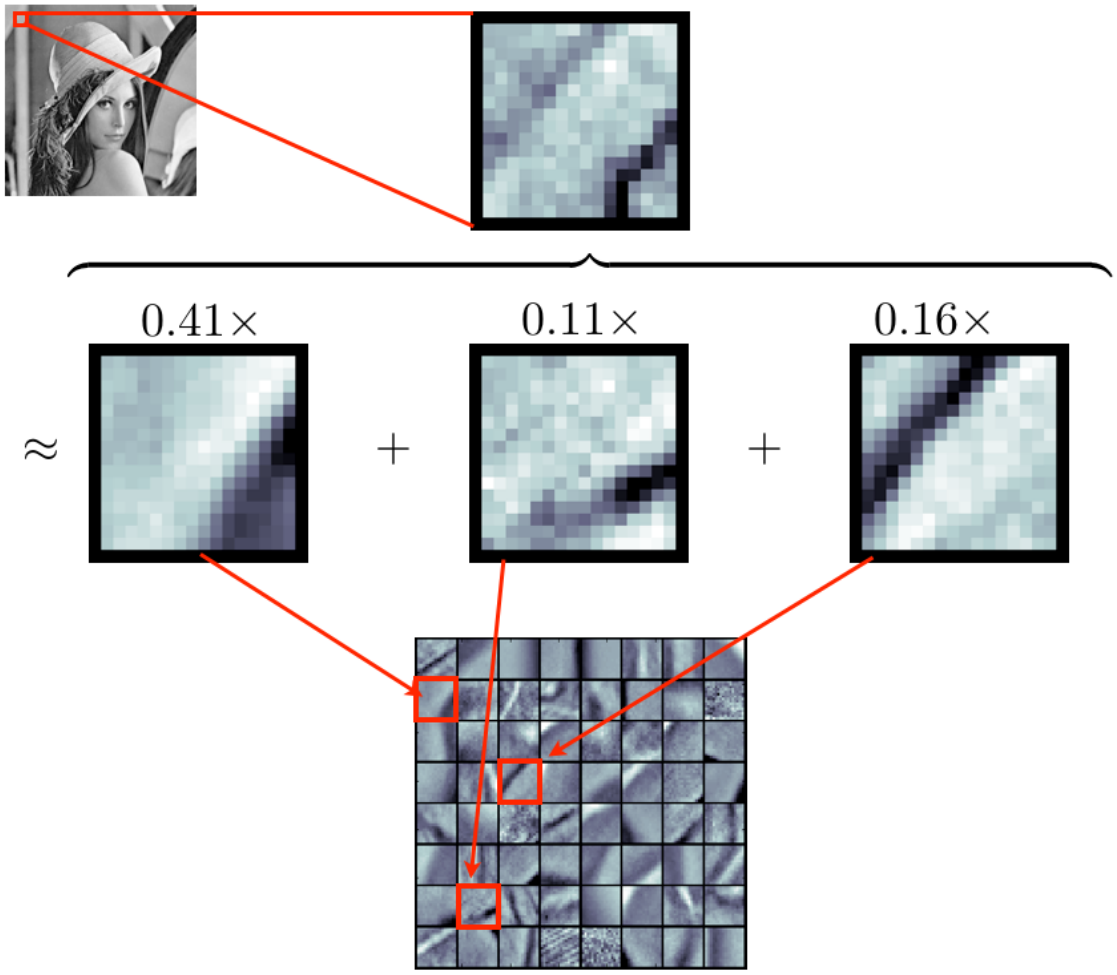

Figure 3.1: Linear combination of an image patch: the 16×16 pixel image patch extracted from the natural image in the top left corner is approximated as the linear combination of three elements out of the 64 dimensional codebook.

Using only this three basis elements of course gives only a rough approximation of the input patch and one could add more basis patches to get a better approximation. However the whole idea behind sparse coding is to only take as few basis patches as possible to approximate the input patch as good as possible. Clearly there has to exist a tradeoff between approximation accuracy by using more input patches and using as less patches as possible and therefore loose

some accuracy. This is also visible in the formulation as optimization problem in formula 3.57.

One question that arrises is whether it is favorable to use many basis elements to compute a good approximation or to use only a few but dominant ones. There exists no global valid answer but recent literature has shown that for various computer vision related tasks sparse representations outperform non-sparse ones [68, 103, 66, 37].

Without any formal details one can easily understand the rationale behind this statement using the following analogue inspired from [99].

One can think of an individual that has the ability to be very good at distinguishing different tastes of fruit juices when he drinks them. He knows apple juice, orange juice, strawberry juice, banana juice and lime juice. These five juices would form the formerly mentioned basis set or dictionary. One can understand that for this individual it is now relatively easy if he has to identify the ingredients of a mixed fruit juice that contains a high percentage of one juice (lets assume 93% apple juice). Put into vector form, this unknown juice could be represented as $(0.93, 0, 0, 0.5, 0)$ if the vector order is (apple,orange,strawberry,banana,lime). Even if this sparse feature vector is not a good representation, as it does not contain 100% of the ingredients, the most dominant information is present.

Contrary it would be quite hard to name all ingredients if the juice contains 20% apple juice, 20% orange juice, 20% strawberry juice, 20% banana juice and 20% lime juice written in vector form as $(0.2, 0.2, 0.2, 0.2, 0.2)$. Even if this second vector is more complete, meaning that the percentage of known ingredients is higher with respect to the first one, it lacks the sparseness property.

In many classification problems this analogue holds as well, as it has been shown that many classification algorithms achieve higher classification rates using sparse feature vectors.

Sparse coding is born out of the idea that sparse codes are used in cortical computations, therefore almost all of the first publications about sparse coding come from the area of biology.

In [74] it has been shown that sparse representations lead to receptive fields similar to the ones in the primary visual cortex of mammalians. With an experiment where the authors learned 192 basis functions from 16×16 pixel patches from natural scenes they could generate spatially localized, oriented receptive fields by imposing only two global objectives on a linear coding scheme: the information to be coded should be preserved and the coefficient activity should be sparse.

In recent years many new theoretical insights and algorithms have been published: In [54] and [35] the authors have focused on creating fast algorithms and approximations to solve

the sparse coding optimization problem while the papers [64, 65] focused on providing fast and efficient online learning algorithms for the creation of the visual codebook. In [20] the author published a comparative study of the encoding capabilities of sparse coding and vector quantization.

Mutch performs object recognition based on sparse coding in [68, 69]; the papers [99, 103] focus on face recognition using sparse features; [37] uses sparse coding for audio classification; [66] describes how to use sparse coding for image restoration (inpainting), and in [80] invariant features are learned using sparse coding.

These papers have shown that, for many tasks it is favorable to combine local low-level features to a mid-level representation of higher complexity. These representations are typically called mid-level because they combine several low-level features together, but remain close to that level, lacking a high-level description of the objects. In recent publications it has become quite popular to use SIFT as low-level descriptor for Sparse Coding, but the method is generally independent from the used low-level representation. In [19] HOG has been selected as low-level feature descriptor for the task of image classification, but other features like HOW are also possible. There the codebook would encode motion primitives, which are combined to describe a new motion scene.

It is to note, that Sparse Coding is not the only possibility to create mid-level feature representations, in fact one such method, namely the bag-of-words-model, has already been introduced in chapter 2.2. Representing low-level features as a histogram distribution, in the case of the bag-of-words-model, already creates a higher-level feature representation with different characteristics. Even though it neglects local information of the low-level features, recent publications have shown good results. Other methods to generate mid-level representations include spatial pyramids [50] and deep belief networks [79, 55].

3.2 Formal Definition

As the goal of this chapter is to formally define sparse coding as a convex optimization problem, a general introduction and overview of numerical optimization and some important concepts useful for the understanding of the sparse coding definition is given. Two special classes of numerical optimization, *least squares* and *convex optimization*, are described.

Thereafter the problem of matrix factorization or matrix decomposition, how it is also called, is shortly discussed, as it builds a conceptual basis for many algorithms like sparse coding. A special case of matrix factorization, called non-negative matrix factorization, is then introduced, motivated and fundamental algorithms are discussed. Based on that knowledge sparse coding is defined and algorithms to solve this convex optimization problem are reviewed.

3.2.1 A Short Introduction to Numerical Optimization

Formally, a (continuous) numerical or mathematical optimization problem can be written in its standard form as

$$\begin{aligned} &\text{minimize } f_0(x) \\ &\text{subject to } f_i(x) \leqslant b_i, i = 1 \dots m \end{aligned} \tag{3.1}$$

, where the vector x denotes the n-dimensional optimization variable and $f_0 : \mathbf{R}^n \to \mathbf{R}$ is called the objective function. The functions $f_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 1 \dots m$ are called (inequality) constraint functions and $b_{1 \cdots m}$ are the bounds for these constraints. The solution for this problem is the vector with the smallest function value for the objective function from all vectors that satisfy the constraints. Formally this means that for any y with $f_1(y) \leqslant b_1 \dots f_m(y) \leqslant b_m \to f_0(y) \leqslant f_0(x^*)$.

Generally various such optimization problems are grouped into classes of optimization problems based on the form of the objective function or its constraints. As an example, the optimization problem defined in Eq.(3.1) is called linear optimization problem if all functions (objective and constraints) f_i for $i = 0 \dots m$ are linear and therefore satisfy

$$\begin{aligned} &f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y) \\ &\text{for all } x, y \in \mathbf{R}^n \text{ and } \alpha, \beta \in \mathbf{R} \end{aligned} \tag{3.2}$$

. As an exemplary application of such optimization problems, portfolio optimization could be mentioned. The aim of portfolio optimization is to spend a capital into a set of i assets. x_i contains the investment into a single asset and therefore the vector x describes an individual portfolio. The constraints could represent restrictions on the budget (invest no more than 5000$), restrictions on individual assets (asset 3 can not be bought in combination with asset 6 or investment on asset 5 must be higher than 400$) or other requirements (no negative amount of money can be invested). The objective function could be a measure of the overall risk of the portfolio return. Then solving the Eq.(3.1) in this setting enables to automatically choose the best combination of assets that minimize risk based on all possible asset combinations that satisfy the constraints.

Least-squares problem

The so-called least squares problem describes a special class of optimization problems. All problems in this class are constraint-free (meaning that $m = 0$) and the objective function is a sum of terms of the form $a_i^T x - b_i t$. The optimization problem therefore can be described as

$$\text{minimize } f_0(x) = ||Ax - b||^2 = \sum_{i=1}^{k} (a_i^T x - b_i t)^2 \tag{3.3}$$

. Again $x \in \mathbf{R}$ is the optimization variable, $A \in \mathbf{R}^{k \times n}$ and a_i^T is a single row of A. Note that in this setting $k \geqslant n$. It is possible to reduce the solving of a least-squares problem to solving a set of linear equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ &\cdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n &= b_m \end{aligned} \tag{3.4}$$

$$\tag{3.5}$$

, which can also be represented in matrix notation as

$$Ax = b \tag{3.6}$$

. Left-multiplying by A^T gives

$$(A^T A)x = A^T b \tag{3.7}$$

, for which the analytical solution

$$x = (A^T A)^{-1} A^T b \tag{3.8}$$

exists. There exist good algorithms to solve such least-squares problems and it is also possible to take into account some special structures within the coefficient matrix A. For example if A is sparse, meaning that there are less than $k * n$ non-zero entries within A the least-squares problem can usually be solved much faster. To identify a given problem as least-squares optimization problem one needs just to ensure that the optimization function is a quadratic function. There exist various extensions of the least-squares problem by modifying the cost function with the goal to penalize the influence of single terms to the final solution. One example is weighted least squares:

$$\text{minimize } f_0(x) = \sum_{i=1}^{k} w_i (a_i^T x - b_i t)^2 \tag{3.9}$$

, where $w_1 \dots w_n$ are positive weights chosen to embed some knowledge about the size of individual terms $a_i^T - b_1$. In regularized least squares extra terms are added to the original cost function to induce some prior knowledge and create a tradeoff between minimizing the original objective function and the introduced balancing term. If a positive multiple ϕ of the sum of squared variables is added, the minimization problem becomes

$$\text{minimize } f_0(x) = \sum_{i=1}^{k} (a_i^T x - b_i t)^2 + \phi \sum_{i=1}^{n} x_i^2 \tag{3.10}$$

and penalizes large values of x which could be desired given some prior knowledge about the optimization problem.

Convex Optimization

A convex optimization problem, which optimizes convex functions over convex sets, can be expressed in the general form

$$\begin{aligned} &\text{minimize } f_0(x) \\ &\text{subject to } f_i(x) \leqslant b_i, i = 1 \dots m \end{aligned} \tag{3.11}$$

, where the optimization function f_0 and the optimization constraints $f_1, \dots, f_m$ satisfy

$$\begin{aligned} &f_i(\alpha x + \beta y) \leqslant \alpha f_i(x) + \beta f_i(y) \\ &\text{for all } x, y \in \mathbf{R}^n \text{ and } \alpha, \beta \in \mathbf{R},\ \alpha + \beta = 1,\ \alpha \geqslant 0, \beta \geqslant 0. \end{aligned} \tag{3.12}$$

. Both, the introduced least squares problem in Eq.(3.3) and the linear optimization problem in Eq.(3.2), are special cases of the convex optimization problem. When comparing Eq.(3.12) with Eq.(3.2), we see that equality is replaced by inequality, which means that convexity is a more general restriction than linearity and therefore any linear optimization can be seen as a specialization of the convex optimization idea.

3.2.2 Matrix Factorization

Matrix factorization [31] is the right-hand product of the following approximation:

$$X \approx A_1 A_2 \dots A_k \tag{3.13}$$

for an input matrix X, where $A_1, A_2, \dots A_k$ are the factor matrices and k is the number of such factor matrices. Or put in other words matrix factorization is the approximation of a single matrix as a product of two or more matrices. For most problem settings $k = 2$ or $k = 3$, for the reminder of this chapter matrix factorization is discussed for the case of $k = 2$ and the standard notation

$$X \approx WH \tag{3.14}$$

is used. W is often called *mixing matrix* containing the decomposition features column-wise while H contains the activations for each feature on input matrix X. This formulation already shows the relation between matrix factorization and the short informal introduction to Sparse Coding given in the previous section. Recapitulating, Sparse Coding describes a process to represent input data as sparse linear combination of a basis set. This is exactly what the matrix factorization states: input X is described in terms of (generally non-sparse) activations H of a basis set W. Therefore in the remainder of this chapter the general idea of matrix factorization is further developed to incorporate sparseness and generate only non-negative activations, to finally arrive to the formal definition of Sparse Coding.

Generally, matrix factorization is non-unique meaning that there exists more than one solution for a given X, and there exist well-known algorithms to solve that problem, like LU-decomposition where X is decomposed into a lower-triangular and an upper-triangular

matrix, QR-decomposition where one of the two factor matrices is orthogonal and the other one right triangular which are described in more detail based on [32]. Other methods to solve this problem include singular value decomposition, Cholesky decomposition or the Jordan decomposition method.

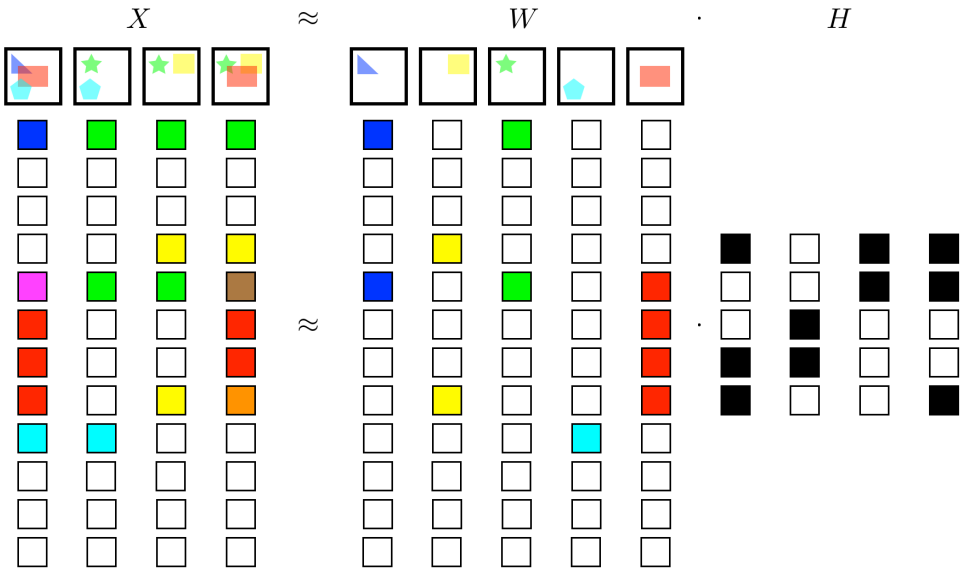

Figure 3.2: Illustration of a general matrix factorization: X contains the column-wise features, W is a basis matrix and H contains the activations of each basis for a given feature.

LU Decomposition

The general idea of LU decomposition is to decompose $X = L \cdot U$ using a modification of Gaussian elimination, like the iterative Doolittle algorithm which will be described briefly in the following. Note that L denotes a lower, and U an upper triangular matrix.

Assume a quadratic $N \times N$ matrix X, with non-zero entries on the diagonal. In a first step, the initial matrix to be decomposed is taken as $X^{(0)} = X$. The following procedure is then repeated for $n = 1, \ldots, N-1$.

The entries of the n^{th} column of X below the diagonal are eliminated. This means: each each entry in this column below the diagonal, i.e. $X_{i,n}$ for $i = n+1, \ldots, N$. To eliminate these entries, the idea is to use

$$X_{i,n} = -\left(\frac{X_{i,n}^{(n-1)}}{X_{n,n}^{(n-1)}}\right) \cdot X_{n,n}^{(n-1)} \tag{3.15}$$

. The current lower triangular matrix is defined as the identity matrix with one "modified" column:

$$L_{i,n}^{\prime(n)} = \begin{cases} -\frac{X_{i,n}^{(n-1)}}{X_{n,n}^{(n-1)}} & \text{if } i > n, \\ 0 & \text{otherwise.} \end{cases} \tag{3.16}$$

$$L^{(n)} = I + L^{\prime(n)} \tag{3.17}$$

. The matrix is then updated:

$$X^{(n)} = L^{(n)} \cdot X^{(n-1)} \tag{3.18}$$

. This process is repeated until $n = N - 1$. After that, in the matrix $X^{(N-1)}$, all entries below the diagonal are deleted. Thus it represents the upper triangular matrix from the definition:

$$U = X^{(N-1)} \tag{3.19}$$

. Finally, one can bring everything together:

$$\begin{aligned} X &= L^{(1)^{-1}} \cdot L^{(1)} \cdot X^{(0)} \\ &= L^{(1)^{-1}} \cdot X^{(1)} \\ &= L^{(1)^{-1}} \cdot L^{(2)^{-1}} \cdot L^{(2)} \cdot X^{(1)} \\ &= L^{(1)^{-1}} \cdot L^{(2)^{-1}} \cdot X^{(2)} \\ &= \dots \\ &= L^{(1)^{-1}} \cdot \dots \cdot L^{(N-1)^{-1}} \cdot X^{(N-1)} \end{aligned} \tag{3.20}$$

. Thus the lower triangular matrix is defined as

$$L = L^{(1)^{-1}} \cdot \dots \cdot L^{(N-1)^{-1}} \tag{3.21}$$

and from this one gets

$$X = L \cdot U \tag{3.22}$$

. A more generalized definition of the LU decomposition also allows for zero-entries on the diagonal, where the idea is to exchange the rows inside matrix, i.e. to apply some permutation, such that the problem can be overcome, leading to

$$P^{-1} \cdot X = L \cdot U \tag{3.23}$$

QR Decomposition

The principle of the QR decomposition is to factorize a $N \times M$ matrix A according to

$$X = Q \cdot R \tag{3.24}$$

, where Q is an orthogonal matrix and R an upper triangular matrix.

This method of factorization can be applied to linear least squares problems

For decomposing a matrix X in this way, there exist several approaches. One of them, the Gram-Schmidt process is shortly covered in the following:

The matrix X is treated column-wise, knowing that $X = [X_1, \ldots, X_N]$. A projection on the column vectors, denoted as x, is then defined as

$$\text{proj}_e x = \frac{e^\top x}{e^\top e} \cdot e \tag{3.25}$$

. From this one further defines

$$\begin{aligned}
u_1 &= x_1 & e_1 &= \frac{u_1}{\|u_1\|} \\
u_2 &= x_2 - \text{proj}_{e_1} x_2 & e_2 &= \frac{u_2}{\|u_2\|} \\
u_3 &= x_3 - \text{proj}_{e_1} x_3 - \text{proj}_{e_2} x_3 & e_3 &= \frac{u_3}{\|u_3\|} \\
&\vdots & &\vdots \\
u_K &= x_K - \sum_{k=1}^{K-1} \text{proj}_{e_k} x_K & e_K &= \frac{u_K}{\|u_K\|}
\end{aligned} \tag{3.26}$$

Applications of Matrix Factorization

Based on the quality measures used for the approximation and the constraints imposed on the factor matrices this factorization builds the basis for many important techniques.

Exemplary for other applications the use of matrix factorization as basis for recommender systems is described based on the publications [76, 48, 89].

Generally, recommender systems are information processing engines aiming to predict the preference or rating of an item not yet rated by a user based on item caracteristics (content-based recommender systems) or user models (collaborative filtering recommender systems). In a collaborative filtering system users are defined by the set $U = \{u_1, u_2, \ldots, u_p\}$ and items I are expressed as $I = \{i_1, i_2, \ldots, i_q\}$ The goal now is to compute a rating matrix R containing

all rated items i_q for all users u_p. Based on a training matrix $\hat{R}$ containing ratings for a subset of users and items the goal is to decompose it into two individual matrices:

$$R \approx UI = \hat{R} \tag{3.27}$$

, where matrix U contains the users and each item is described as vector in I. In case of a tourist recommender system one such vector in I could contain the hotel category and the corresponding entry in U determines how much the user likes that specific hotel category. Single recommendations can easily be computed with the dot product of the feature vector from user u_i and item i_j.

3.2.3 Non-Negative Matrix Factorization

Since most physical quantities, like amounts, lengths or weights, are non-negative and the input matrix X for a matrix decomposition can contain such information, it seems rational to extend the matrix factorization formulation and imposing a non-negativity constraint on the factor matrices. This extension has first been published as *curve resolution* [56] and *positive matrix factorization* but is now widely known as *non-negative matrix factorization* (NMF) [52].

To measure the quality of the approximation, one has to define a cost function that measures the distance between two of the (non-negative) factor matrices. A popular choice (also taken by D. Lee in [53]) is to take the Euclidean distance between W and H:

$$\|W - H\|^2 = \sum_{i,j} (w_{i,j} - h_{i,j})^2 \tag{3.28}$$

. Using this cost function, it is now possible to formulate the non-negative matrix factorization as optimization problem:

$$\begin{aligned} &\min_{W,H} \|X - WH\|^2 \\ &\text{subject to } W, H \geq 0 \end{aligned} \tag{3.29}$$

. Here $W \geq 0$ and $H \geq 0$ are the key part as they impose the non-negativity constraint and are the main difference between ordinary matrix factorization and non-negative factorization. Given a non-negative matrix X with m columns containing a n-dimensional data vector each it is then approximately factorized into the $n \times r$ matrix W and $r \times m$ matrix H where r can be chosen arbitrarily but is often smaller than m or n resulting in a compression of the data contained in X. Eq.(3.14) can be written for each input vector (column) separately as

$$\begin{aligned} x_1 &\approx W h_1 \\ x_2 &\approx W h_2 \\ x_m &\approx W h_m \end{aligned} \tag{3.30}$$

. From this formulation it becomes easy to see that each input data vector x_i is approximated as a linear combination of a common basis matrix W weighted by h. Computing W can therefore be interpreted as learning a good basis optimized for linear approximations of X.

The function $\|X - WH\|^2$ is convex in W holding H fixed and in H holding W fixed, but is not convex in the general case for both variables. Therefore, most algorithms alternate between the two convex optimization problems when solving the factorization, instead of solving a non-convex optimization problem.

Algorithms

In the following different algorithms for solving the optimization problem defined in 3.29 are described.

Gradient descent algorithm

A simple method to compute W and H is to use an alternating gradient descent method. Provided that the gradient of the optimization function $f(x)$ can be computed gradient descent finds the nearest local minimum from a starting point p_0 by iteratively moving from p_i to p_{i+1} into the direction of $-\nabla f(p_{i)})$ since this is the direction where $f(x)$ decreases fastest. Alternating in this context means that gradient descent is alternatingly applied to H and W. The general descent algorithm is described in algorithm 1, for gradient descent the descent direction $\triangle x = -\nabla f(x)$ is used.

Algorithm 1 General descent algorithm

```
given a start point x

repeat
    determine a descent direction △x

    choose a step size t > 0 (line search)
    x = x + t△x
until convergence
```

In the following a simple gradient descent update rule is developed based on [21] and [8]. Considering the already mentioned Euclidean cost function as approximation error gives

$$J_E = \|X - WH\|^2 = \sum_{i,j}(x_{i,j} - [WH]_{i,j})^2 \tag{3.31}$$

. For one gradient descent step we update H according to

$$H = H - \nabla \frac{\partial J_E}{\partial H} \tag{3.32}$$

or expressed as individual terms:

$$h_{ki} = h_{ki} - \nabla_{ki} \frac{\partial J_E}{\partial h_{ki}} \tag{3.33}$$

. Expressing the cost function in matrix notation gives

$$J_E = \|X - WH\|^2 = \frac{1}{2} tr((X - WH)^T (X - WH)) \tag{3.34}$$

, where the *trace* function $tr()$ of a matrix is the sum of its main diagonal elements defined as

$$tr(A) = a_{11} + a_{22} + \cdots + a_{nn} = \sum_{i=1}^{n} a_{ii} \tag{3.35}$$

. Differentiating J_E with respect to ∂H gives

$$\begin{aligned} \partial J_E &= -tr((X - WH)^T W \partial H) && (3.36)\\ &= -tr(W^T X - W^T WH)^T \partial H) && (3.37)\\ &= -\sum_{ki} \left[W^T X - W^T WH\right]_{ki} \partial s_{ki} && (3.38) \end{aligned}$$

and therefore one gets

$$\frac{\partial J_E}{\partial s_{ki}} = -\frac{\left[W^T X - W^T WH\right]_{ki} \partial s_{ki}}{\partial s_{ki}} = -(\left[W^T X\right]_{ki} - \left[W^T WH\right]_{ki}) \tag{3.39}$$

. . Therefore the final update rule for H becomes

$$h_{ki} = h_{ki} + \nabla_{ki}(\left[W^T X\right]_{ki} - \left[W^T WH\right]_{ki}) \tag{3.40}$$

. Since W and H are symmetric, the same calculations will result in the gradient update step for W:

$$w_{ki} = w_{ki} + \nabla_{ki}(\left[XH^T\right]_{pn} - \left[WHH^T\right]_{pn}) \tag{3.41}$$

. The simple gradient descent algorithm to solve the NMF problem then alternates between the update steps in Eq.(3.40) and Eq.(3.41), until some convergence criterion like a fixed maximum number of iterations is met. This procedure is denoted in algorithm 2. Since the two derived update rules do not maintain the positivity contains on the elements within W and H, one would use the following update rules to enforce the non-negativity:

$$w_{ki} = max(0, w_{ki} + \nabla_{ki}(\left[XH^T\right]_{pn} - \left[WHH^T\right]_{pn})) \tag{3.42}$$

$$h_{ki} = max(0, h_{ki} + \nabla_{ki}(\left[W^T X\right]_{ki} - \left[W^T WH\right]_{ki})) \tag{3.43}$$

. Like for all other gradient descent applications the crucial parameter to tune is the step size which also highly influences the final convergence of the algorithm. Furthermore enforcing the non-negativity after each update step by setting all negative values to the smallest positive value 0 makes convergence analysis even harder. As mentioned in [8], using a simple geometric update rule for the step size, like scaling by a fraction at each iteration often produces poor factorizations and is therefore very sensitive to the initialization of W and H.

Algorithm 2 Gradient descent algorithm for NMF

1: initialize W and H randomly
2: $W = rand(m, k)$,
3: $H = rand(k, n)$
4: **repeat**

$$W = W - \nabla_W \frac{\partial f}{\partial W}$$
$$H = H - \nabla_H \frac{\partial f}{\partial H}$$

5: **until** convergence

Multiplicative Update Algorithm

In [53] Lee proposed the following multiplicative update rules to be non-increasing under the Euclidean distance measure:

$$H = H \frac{(W^T X)}{(W^T W H)} \tag{3.44}$$
$$W = W \frac{(X H^T)}{(W H H^T)} \tag{3.45}$$

. In practice, a small constant is added to the denominator to avoid division by zero:

$$H = H \frac{(W^T X)}{(W^T W H + 10^{-9})} \tag{3.46}$$
$$W = W \frac{(X H^T)}{(W H H^T + 10^{-9})} \tag{3.47}$$

. The most important difference to the already mentioned gradient descent update is that for the multiplicative update each update consists of a multiplication with a factor while in gradient descent a addition is used at each iteration. These multiplicative update rules can be derived from the already mentioned gradient descent update rules in Eq.(3.40) and Eq.(3.41) using a clever choice of ∇_{ki}. If it is chosen such that the first and third terms in Eq.(3.40) and Eq.(3.41) vanish, the already presented multiplicative update rules can be constructed.

Choosing $\nabla_{ki} = \frac{h_{ki}}{[W^T W H]_{ki}}$ gives

$$h_{ki} = h_{ki} + \nabla_{ki}(\left[W^T X\right]_{ki} - \left[W^T W H\right]_{ki}) \tag{3.48}$$

$$= h_{ki} + \frac{h_{ki}}{[W^T W H]_{ki}}(\left[W^T X\right]_{ki} - \left[W^T W H\right]_{ki}) \tag{3.49}$$

$$= h_{ki}(1 + \frac{\left[W^T X\right]_{ki}}{[W^T W H]_{ki}} - 1) \tag{3.50}$$

$$= h_{ki}\frac{\left[W^T X\right]_{ki}}{[W^T W H]_{ki}}) \tag{3.51}$$

and using $\nabla_{ki} = \frac{h_{ki}}{[W H H^T]_{ki}}$, the same steps are used to derive the update rule for W:

$$w_{ki} = w_{ki} + \nabla_{ki}(\left[X H^T\right]_{pn} - \left[W H H^T\right]_{pn}) \tag{3.52}$$

$$= w_{ki} + \frac{h_{ki}}{[W H H^T]_{ki}}(\left[X H^T\right]_{pn} - \left[W H H^T\right]_{pn}) \tag{3.53}$$

$$= w_{ki}(1 + \frac{\left[X H^T\right]_{ki}}{[W H H^T]_{ki}} - 1) \tag{3.54}$$

$$= w_{ki}\frac{\left[X H^T\right]_{ki}}{[W H H^T]_{ki}}) \tag{3.55}$$

Algorithm 3 Multiplicative update algorithm for NMF

1: initialize W and H randomly
2: $W = rand(m, k)$,
3: $H = rand(k, n)$
4: **repeat**

$$H = H\frac{(W^T X)}{(W^T W H + 10^{-9})}$$
$$W = W\frac{(X H^T)}{(W H H^T + 10^{-9})}$$

5: **until** convergence

It has to be noted that, although this is the first well-known algorithm to solve the problem of non-negative matrix factorization and has therefore served as a baseline for newer algorithms this method has some disadvantages. First of all it has been shown that the multiplicative update rule needs far more steps to converge, compared to the two other methods, steepest descent and alternating least squares. A more serious problem however is that the proof of convergence to a local minimum initially claimed by Lee and Seung has been shown to not hold in [33] and other publications.

Also this non-negative matrix factorization has been extended: by using the squared Frobenius norm as a penalty term "smoothness" can be enforced as it penalizes large values within the matrix.

Another interesting extension to the NMF algorithm has been proposed in [95], where a temporal continuity objective is used for the task of sound source separation.

The objective important to arrive to the concept of sparse coding is a sparsity constraint on the non-negative matrix factorization formulated by P. Hoyer in 2002 [40] which is discussed in the next chapter. The extension from the general matrix factorization to a non-negative matrix factorization has already been motivated in the introduction to this section. In addition, from a biologically inspired point of view, since Sparse Coding (and therefore also non-negative matrix factorization) can be related to the neuronal activity in the brain, this extension makes sense, because the neurons firing intensity can not be negative. Furthermore, if negative activations are allowed, two activations could cancel each other out and a more compact, sparse representation could be achieved without the canceling activations.

The question why one should also consider a sparsity extension to the NMF formalization has been answered in the experiments conducted in [40]. The author constructed 10 individual features and used them to create random data vectors. They then performed non-negative Sparse Coding and non-negative Matrix factorization on this data and analyzed if both algorithms can effectively identify and reconstruct the basis features. The results show that NMF cannot represent all hidden features with any given dimensionality. With a specific number of components NMF can extract all hidden components resulting in zero reconstruction error, and there exists no overcomplete representation to improve this result. When using nonnegative Sparse Coding however, combinations of the original features are learned, resulting in a sparser representation of the input patterns.

3.2.4 Sparse Coding

To incorporate sparsity into the optimization problem presented in Eq.(3.29) a sparsity penalty function is added to the Euclidean distance cost function giving the following optimization problem:

$$\begin{aligned} &\min_{W,H} \|X - WH\|^2 + \lambda S(W) \\ &\text{subject to } W, H \geq 0 \end{aligned} \tag{3.56}$$

, where $\lambda \geq 0$ is the sparsity weight and $S()$ is the sparsity penalty function.

Mapping this optimization problem into the sparse coding context, the following notation is used from now on: $X = [x_1, x_2, \ldots, x_m] \in \mathbb{R}^{m \times n}$ denotes the unlabeled input data we want to compute the sparse codes for, therefore $x_i \in \mathbb{R}^n$ is one such input vector, $D \in \mathbb{R}^{n \times d}$ is a dictionary/codebook containing d elements. u_i denotes one sparse activation vector given

dictionary D and input vector x_i which we want to compute. Then the construction of sparse codes can be formulated exactly as in Eq.(3.56) with just a slightly different notation:

$$\min_{U,D} \sum_{i=1}^{m} \|x_i - u_i D\|^2 + \lambda S(u_i) \tag{3.57}$$

. As this definition of sparse coding does not impose any non-negativity constraint on U or D, the data will be described as linear combination of additive and subtractive elements. This implies that there exists a small chance that from the few non-zero factors one cancels out another. This however stands in contradiction with the intuition of representing the input data X as a combination of the activation to approximate again the whole data. This restriction seems also plausible from a neuroscience perspective as sparse coding has been created to represent the firing patterns of neurons and these impulses cannot be negative. Including the non-negativity constraints the optimization problem becomes the following:

$$\begin{aligned} &\min_{U,D} \sum_{i=1}^{m} \|x_i - u_i D\|^2 + \lambda S(u_i) \\ &\text{subject to } \forall i, j; U_{i,j} \geq 0, \forall i u_i = \|1\|, \lambda \geq 0 \end{aligned} \tag{3.58}$$

. Again $S(x)$ denotes a sparsity penalty function. In [75] the authors showed that choices for the penalty function like $S(x) = -e^{-x^2}$, $S(x) = log(1 + x^2$, and $S(x) = \|x\|$ all produce similar sparse codes. The typical choice of $S(x) = \|x\|$ however imposes a problem: the objective function can always be decreased by scaling down U and scaling up D because $S(x)$ is strictly increasing on the values x. This behavior comes from the fact that for any $\alpha > 1$ using $D = \alpha D$ and $U = \frac{1}{\alpha} U$ does not alter the first term $\sum_{i=1}^{m} \|x_i - u_i D\|^2$ but always decreases $\lambda S(u_i)$.

This formulation as optimization problem balances two terms:

The first term represents a given input sample x_m as a weighted linear combination of the basis atoms within the codebook and the weights given by the activations u_i and minimizes its reconstruction error

On the other hand the second term minimizes the sparsity penalty, meaning that it forces the activation vector u_i, to contain only few non-zero elements. So there exists a tradeoff between low reconstruction error and a sparse activation vector, since the fewer basis atoms are used to represent the input data, the higher the reconstruction error will be. This tradeoff is controlled by λ.

Feature-Sign Search Algorithm

Since computing large, over complete representations is a computationally challenging task, it is important to have efficient algorithms for solving the sparse coding problem. To give an intuition on how such algorithms work, exemplary the so called feature-sign search algorithm proposed in [54] is discussed as it is currently one of the most efficient ways to solve the optimization problem presented in Eq.(3.56).

The whole procedure is based on the knowledge that the whole sparse coding formulation is not a convex optimization problem, but it can be splitted into two subproblems, which both can be solved as convex optimization problems. Performing alternating optimization on these two subproblems, the initial optimization problem can be solved. Using either one of L_1-penalty ($\|u_j\|_1$), epsilonL_1 penalty ($\sqrt{s_j^2 + \epsilon}$) or log-penalty ($log(1 + s_j^2)$) as sparsity penalty function in Eq.(3.58), the optimization problem becomes convex with respect to D, when the activation-matrix U is fixed and vice-versa.

The authors argue that generic tools to solve the remaining convex optimization problems like gradient-based methods or generic solvers are to slow and therefore propose their algorithm. It is shown that, when holding the activations U fixed and solving the optimization problem for each u_j separately, knowing only the sign of the u_j^i at the optimal value is reduced to the L_1-regularized least squares problem to a unconstrained quadratic optimization problem.

Algorithm 4 Feature-sign search algorithm

initialize $x = \vec{0}, \theta = \vec{0}, \text{set} =$ where $\theta_i \in \{-1, 0, 1\}$
From zero coefficients of x select $i = \arg\max_i |\frac{\partial ||y - Ax||^2}{\partial x_i}|$
Activate x_i (add i to the active set) only if locally improves the objective:
if $|\frac{\partial ||y - Ax||^2}{\partial x_i}| > \rho$ then$\theta_i = -1, \text{set} = \{i\} \bigcup \text{set}$
if $|\frac{\partial ||y - Ax||^2}{\partial x_i}| < -\rho$ then$\theta_i = 1, \text{set} = \{i\} \bigcup \text{set}$
Feature-sign step:
$\hat{A}$ is a sub matrix of A containing only the columns corresponding to the active set
$\hat{x}$ and $\hat{\theta}$ are sub vectors of x and θ corresponding to the active set
compute the analytical solution: $\text{minimize}_{\hat{x}} ||y - \hat{A}\hat{x}||^2 + \rho\hat{\theta}^T\hat{x}$:
$\hat{x}_{new} = (\hat{A}^T\hat{A})^{-1}(\hat{A}^T y - \rho\hat{\theta}/2)$
perform discrete line search on the closed line segment from $\hat{x}$ to $\hat{x}_{\text{new}}$
Check the objective value at $\hat{x}_{\text{new}}$ and all points where any coefficient changes sign
Update $\hat{x}$ (and corresponding entries in x) to the point with the lowest objective value
Remove zero coefficients of $\hat{x}$ from active set and update $\theta = \text{sign}(x)$
Check the optimality conditions:
optimality condition for nonzero coefficients: $|\frac{\partial ||y - Ax||^2}{\partial x_i}| + \rho\text{sign}(x_i) = 0, \forall x_i \neq 0$
optimality condition for zero coefficients: $|\frac{\partial ||y - Ax||^2}{\partial x_i}| \leqslant \rho \forall x_i \neq 0$

The proposed algorithm (also described in pseudo-code in Algorithm 4) tries to "guess" the sign (negative, zero, positive) of the individual activations, hence also the name "feature-sign search". Based on these guesses the remaining unconstrained problem can be solved; refining them at each step until some convergence criterion is met. For the convergence proof of the proposed method or further details the interested reader can refer to the original publication [54].

In addition the same paper contains a second algorithm for solving the remaining optimization problem, the one when holding the dictionary D fixed and solving for the activations U. This reduces to a least squares problem with quadratic constrains, solvable using gradient descent. However the authors propose to solve the problem analytically using the Lagrange dual and show in different experiments that this leads to a huge performance gain.

3.3 Coding Pipeline

The generation of sparse codes can be divided into two major phases, depicted in figure 3.3.

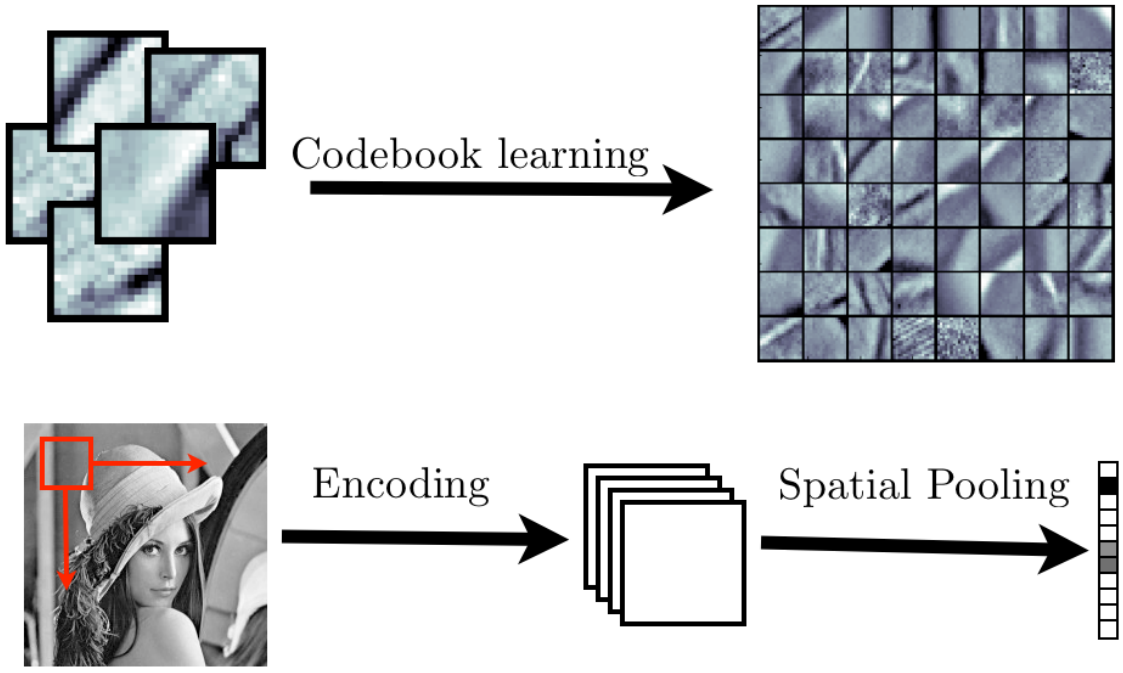

Figure 3.3: Illustration of the sparse coding pipeline: 1) some dictionary learning algorithm is used to create a dictionary from unlabeled input data. 2) The encoding transforms each new input batch into a sparse linear combination of dictionary elements. 3) Spatial pooling is performed to create the final sparse feature descriptor for an input image

In the first phase the fixed basis set, often also referred to as dictionary or codebook, is learned. Given a set of input vectors $(x_1, x_2 \dots x_n) \in \mathbb{R}^n$ an appropriate codebook learning algorithm is used to create the dictionary $D \in \mathbb{R}^{n \times d}$. Details on different codebook learning techniques are explained in section 3.4.

Once the codebook is available the new feature vector x is encoded to create its sparse description. Optionally, but in practice performed often a spatial pooling operation on a set of sparse feature descriptors $s_1, s_2, \dots, s_n$ is performed to generate one single sparse feature descriptor s out of it. In this second phase the sparse representation is computed by minimizing Eq.(3.57) holding the dictionary D fixed.

3.4 Codebook Learning

In the following algorithms to compute or earn the codebook are presented. The first two methods are not special methods for Sparse Coding, but represent general principles useful to generate codebooks. The last method, "Online dictionary learning for Sparse Coding" [64], however represents an algorithm specifically created and tailored for fast online computation of a codebook.

Random Patches

One very basic method to create a codebook is to take d unique random samples from the input data X und use them to construct the d-dimensional codebook. This then means that new data is represented as linear combinations of random data from the dictionary training data and has been shown to result in relatively good sparse codes [20]. One such created codebook is depicted in figure 3.4. Of course this does not involve any sophisticated learning phase as the 'learning' consists of drawing distinct random samples from the input data but for the sake of completeness it is mentioned in this section.

K-Means Clustering

Another more complex choice to create a visual codebook, not necessarily restricted to sparse coding, is the k-means clustering algorithm. K-means is one of the simplest non-hierarchical clustering methods and it is also runtime efficient. The main idea to use it for codebook learning is to group the dictionary training data into k clusters and use the cluster centroids as codebook entries. The pseudo code algorithm to create the k clusters is written in algorithm 5 and can be summarized to the following steps:

- the initial cluster centers $C_{initial}$ are initialized randomly
- each data vector in X is assigned to the cluster whose center of mass is nearest to that vector by means of an appropriate distance function
- for each cluster, its center is recalculated

- the two steps are repeated until some stopping criterion has achieved. This could for example be a fixed number of iterations or, as indicated in the pseudocode until the cluster centers do not change anymore.

Commonly used distance functions are the Euclidean norm $d(x, y) = \|x - y\|$ or the Mahalanobis distance $d(x, y) = \sqrt{(x - y)^T S^{-1}(x - y)}$, where S denotes the covariance matrix.

Algorithm 5 K-Means clustering algorithm

Require: X set of N data vectors, $C_{initial}$ initial k cluster centers

repeat
  $C_{previous} \leftarrow C_{initial}$
  for i=1:N **do**
    P(i) = arg min $d(x_i, c_i)$ ▷ generate new optimal partitions
  end for
  for j=1:k **do**
    C(j) = average of x_i where $P(i) = j$ ▷ generate new optimal centroids
  end for
until $C = C_{previous}$
return C k cluster centroids of X, P cluster label of X

One of the weaknesses of the k-means algorithm, namely that the number of needed clusters needs to be known in advance, does not impose a problem in this setting, as the size of the codebook has anyway to be fixed. For other applications $k \approx \sqrt{\frac{n}{2}}$ is a good heuristic. In figure 3.4 the result of learning a k-means codebook (for $k = 25$) is illustrated.

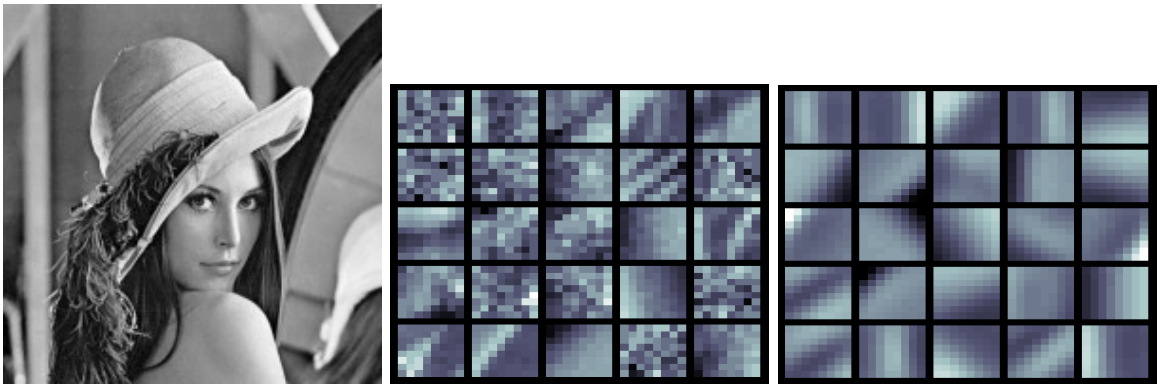

Figure 3.4: Two codebooks generated from 8×8 pixel patches extracted from the first image. The first codebook contains random patches, the second one contains the cluster centers computed with a k-means (for $k = 25$) algorithm.

One major drawback of the original k-means algorithm is that it is defined as batch variant and thus needs all available data during the computation and therefore is quite memory

intensive. Another often mentioned drawback is that the outcoming clusters are affected by the initial positions of the centroids and could give poor results if the optimization loop is not performed until the clusters stabilize. When comparing an exemplary codebook created with the k-means algorithm visible in figure 3.4 one sees that it shares many common patches with a codebook learned with an algorithm specifically invented to be used for sparse coding depicted in figure 3.5.

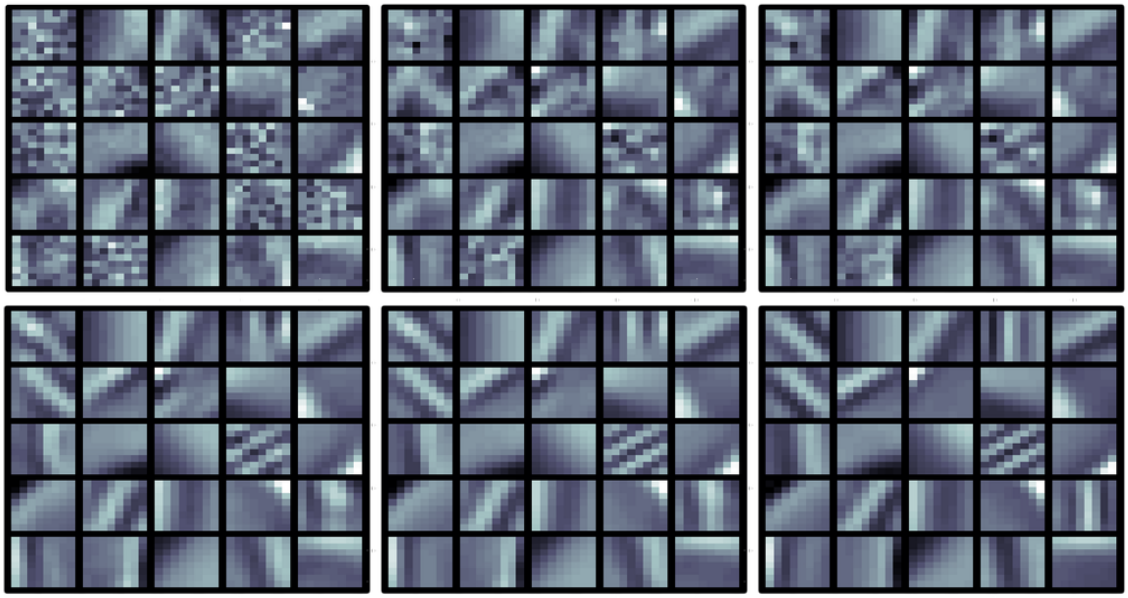

Figure 3.5: Evolution of a codebook learned for time steps $t = \{1, 5, 10, 100, 1000, 10000\}$ based on 8 x 8 pixel patches from a natural image using the Online Dictionary Learning Algorithm for Sparse Coding proposed in [64]

Online Dictionary Learning for Sparse Coding

This section introduces a novel online codebook learning algorithm based on [100, 64, 65]. Online learning in this context and also in general learning problems is a paradigm where one instance is learned at a time. For codebook learning this means that a codebook is constructed based on one data element (a image patch for example) and updated when new data comes in. It is not needed t have all data available at once like in the batch k-means algorithm. Clearly this gives the advantage to compute codebooks for larger datasets that would not fit completely into main memory to be processed by k-means.

The novelty of this algorithm is that as implied already by its name it is an online algorithm to learn the visual codebook by incrementally updating it. Another big advantage is that it is parameter free and it is therefore not needed to tune a learning rate. The pseudocode of the algorithm is summarized in algorithms 6 and 7.

Algorithm 6 Online dictionary learning for sparse coding

Require: $D_0 \in \mathbf{R}^{m \times k}$ initial dictionary, T number of iterations

$A_0 \leftarrow 0$

$B_0 \leftarrow 0$

for $t \leftarrow 1, T$ **do**

Draw random sample x_t from p_x

Sparse coding: compute using LARS

$$\alpha_t \widehat{=} \underset{\alpha \in \mathrm{R}^k}{\operatorname{argmin}} \frac{1}{2}\|x_t - D_{t-1}\alpha\|_2^2 + \lambda\|\alpha\|_1$$

$A_t \leftarrow A_{t-1} + \alpha_t \alpha_t^T$

$B_t \leftarrow B_{t-1} + x_t \alpha_t^T$

Compute D_t using Algorithm 2, with D_{t-1} as warm restart:

$$\begin{aligned} D_t &\widehat{=} \underset{D \in \mathrm{C}}{\operatorname{argmin}} \frac{1}{t} \sum_{i=1}^{t} \frac{1}{2}\|x_i - D\alpha_i\|_2^2 + \lambda\|\alpha_i\|_1 \\ &\widehat{=} \underset{D \in \mathrm{C}}{\operatorname{argmin}} \frac{1}{t}(\frac{1}{2} Tr(D^T D A_t) - Tr(D^T B_t)) \end{aligned}$$

end for

return D_t (learned dictionary)

Algorithm 7 Dictionary Update

Require: $D = [d_1, \ldots, d_k] \in \mathrm{R}^{m \times k}$ input dictionary,

$A = [a_1, \ldots, a_k] \in \mathrm{R}^{k \times k} = \sum_{i=1}^{t} \alpha_i \alpha_i^T$,

$B = [b_1, \ldots, b_k] \in \mathrm{R}^{m \times k} = \sum_{i=1}^{t} x_i \alpha_i^T$

repeat

for $j \leftarrow 1, k$ **do**

update the j-th column to optimize for (9):

$$u_j \leftarrow \frac{1}{A_{jj}}(b_j - Da_j) + d_j$$

$$d_j \leftarrow \frac{1}{max(\|u_j\|_2, 1)} u_j$$

end for

until convergence

return D (updated dictionary)

3.5 Spatial Pooling

After the computation of the sparse codes, one ends up with a set of sparse feature descriptors, each one being a vector. This is also illustrated in figure 3.6. The large colored box on the left is the set of feature vectors, each row corresponds to one sparse descriptor (one is highlighted in red) and each column is the activation of all feature descriptors to one codebook entry. If the sparse codes are computed based on image patches, this matrix can be interpreted as each row being the activation of each single patch, extracted from one image to one specific codebook entry. A column on the contrary represents all (sparse) activations from the codebook to reconstruct the single image patch taken from the whole image. Dark blue indicates zero activation whereas red indicates high activation; therefore one can see that each feature vector and the total matrix are indeed sparse.

The question now is how to combine the information present in the descriptors into one single descriptor to describe the whole image, not only a single patch.

Exactly for this purpose the spatial pooling function comes in. The idea is to combine the activations for one codebook entry into one single value resulting in a feature descriptor having the same size of the codebook, one entry per codebook. One idea could be to take the average of all activations, others could include to take the maximum or minimum activation value, or simply draw a random activation value to express the activity of the whole image with respect to one basis element.

The aim of a spatial pooling operation is therefore, to describe the neighboring feature descriptors by summarizing their characteristics within a certain region of interest. If this region of interest is the whole input image, pooling allows to describe the image as a combination of the local descriptors, resulting in a much more compact representation. A compact representation is favorable for many tasks, but this is not the only reason for feature pooling. When combining these local descriptors, a certain degree of robustness to noise and clutter within single local features can be achieved. Depending on the used pooling function, the resulting descriptor can also be more robust to image transformations then the original local image descriptors.

Feature pooling is not a concept specific to Sparse Coding, but an ingredient for many visual systems. Biologically inspired systems [83, 79, 41, 51] usually use maximum or average pooling, meaning that they construct the global descriptor as maximum or average of the local descriptors. But also popular feature descriptors like HOG or SIFT perform pooling, when combining the local gradient orientations within a predefined neighborhood into a histogram. All pooling applications share the common goal to combine the local features by preserving important information and remove unneeded details. Most of the pooling functions tend to remove spatial information, assuming that to describe an object it is sufficient to know whether a local feature is present or absent. In the following experiments, different pooling functions are therefore evaluated on the same task, in order to evaluate their expressiveness of the local features.

Figure 3.6 shows the feature descriptors for different pooling strategies. The colored feature vectors on the right of the image are the result of different pooling strategies. The first feature vector for instance is the result of the maximum pooling function, meaning that for every codebook entry the maximal activation is used. The following section 3.6 contains the results for a classification task using different pooling strategies. There also the different pooling methods and resulting feature vectors are discussed in more detail.

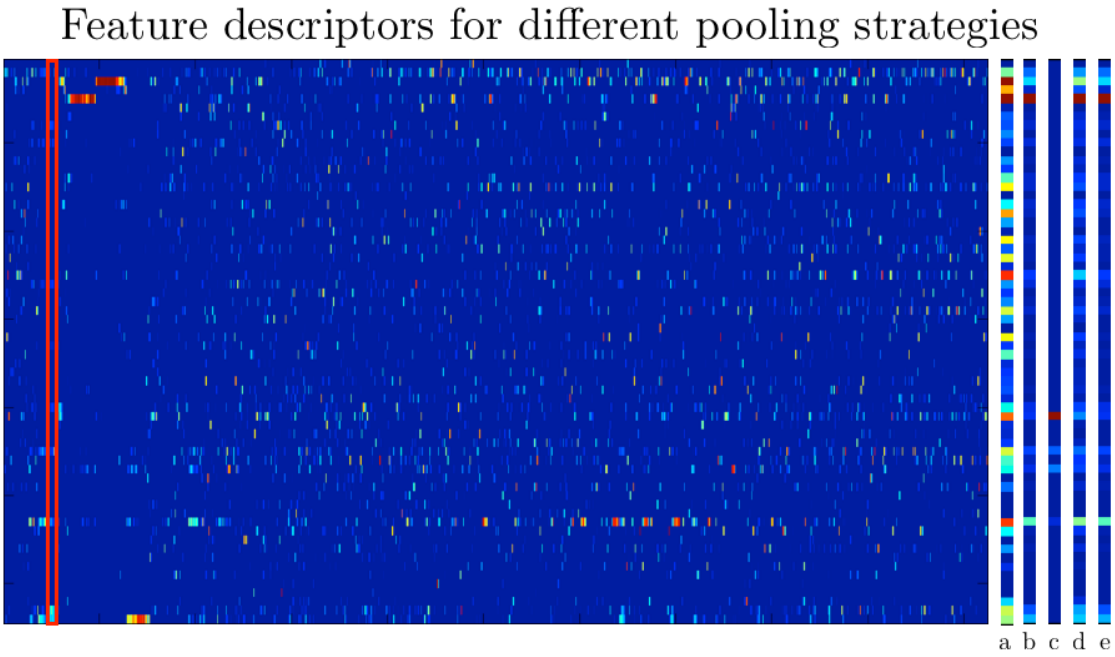

Figure 3.6: All activations intensities on a codebook containing 64 elements. One sparse code for an image patch corresponds to one column (marked in red) in the matrix. On the right side, the resulting feature vectors after the following pooling methods are plotted: (a) maximum pooling, (b) average pooling, (c) random pooling, (d) square root pooling, (e) absolute pooling. Blue indicates 0 activation, red indicates maximum activation.

3.6 Experiments and Results

3.6.1 General Information

This section provides information about general concepts, techniques and methodologies applied in the following experiments. It contains information about Support Vector Machines, a powerful classification method and highlights some characteristics of the used datasets useful for further interpretation of the simulation results.

All experiments have in common that they measure the performance when varying different Sparse Coding parameters, like the codebook size or the pooling method, for a supervised classification task. Generally, solving a classification task requires an algorithm that decides for a given input (e.g. an image) represented as feature vector x to which of the available classes $\mathcal{C}_k$ for $k = 1, \ldots, n$ it most likely belongs, based on previously analyzed labeled samples. This classification algorithm generates an output y, taking one of the possible values $1, \ldots, k$, for each input x, and therefore classifies each x into a class $\mathcal{C}_k$. This can also be written as a mapping function of the form

$$y_k = y_k(x; w) \tag{3.59}$$

, where w denotes a parameter vector for this mapping function. The classification algorithm used to learn this mapping function for the following experiments is called Support Vector Machine and is shortly described in this section.

Support Vector Machines

Support vector machines (SVMs) are based on the idea that linearly non separable data $\boldsymbol{x}$ can be separated by a so-called hyperplane after transferring the data to a higher dimension, using a nonlinear mapping $\varphi()$ [25]. This mapping $\varphi()$ is applied to any training pattern $\boldsymbol{x}_k$ for $k = 1, \ldots, n$, giving

$$\boldsymbol{y}_k = \varphi(\boldsymbol{x}_k) \tag{3.60}$$

A linear discriminant (which represents the hyperplane, separating the high-dimensional data) is then defined as

$$g(\boldsymbol{y}) = \boldsymbol{a}^t \boldsymbol{y} \tag{3.61}$$

where $\boldsymbol{a}^t$ is some weight vector, which has been mapped to the same high-dimensional space as the patterns.

Additionally it has to be defined to which class, in a two-class classification problem ω_1 or ω_2, a pattern belongs. To do so, for each pattern a variable z_k is introduced, taking a value $z_k \in \{-1; +1\}$ depending on the class.

The separating hyperplane can then be linearly mapped to the base axis, ensuring for all the n patterns in this way that

$$z_k \cdot g(\boldsymbol{y}_k) \geq 1 \tag{3.62}$$

Training the SVM consists in finding this hyperplane, or more precisely in finding the (optimal) hyperplane with the highest distance b from the nearest training data, which are also called support vectors, assuming that a higher margin between optimal hyperplane and support vectors means a higher difference between the clusters, and therefore a more general classifier.

The distance b' from any (transformed) pattern $\boldsymbol{y}$ to the optimal hyperplane is given as

$$\frac{|g(\boldsymbol{y})|}{\|\boldsymbol{a}\|} = b' \tag{3.63}$$

When now looking at 3.62, the abolute value is not required any more (as the value is > 1 and thus positive in any case). From this follows, that the distance of any pattern to the optimal hyperplane is at least b:

$$\frac{g(z_k \cdot \boldsymbol{y}_k)}{\|\boldsymbol{a}\|} \geq b \tag{3.64}$$

Thus the real purpose of training or learning is to find a parameter vector $\boldsymbol{a}$ which should maximize this minimal distance b.

Datasets

Caltech 101

Caltech101 is a famous image dataset provided by the California Institute of Technology since 2003. It is composed of a total of 9144, images grouped in 101 different object categories and an additional background category. On average each object category contains between 31 and 800 objects with a resolution of about 300 x 200 pixels. First results on this dataset have been published by Li Fei-Fei using a bayesian algorithm to learn generative visual models in [26]. State of the art performance on this dataset is reported in [101, 12, 11, 91, 58] and ranges between 79% and 84.3% correct classification accuracy using 30 training samples per class.

Caltech 256

Caltech256 [36] is the extension of the Caltech101 dataset, containing 30607 images contained in 256 object categories, again containing also an additional background category. Similar to Caltech101, the Caltech256 dataset has been produced using Google Images search and manually screening and categorizing the images. Reported performance on Caltech256 is lower than on Caltech101, with good results for 30 training samples per class, reaching between 35% and 45% classification rate [12, 11, 102, 96]. The differences between the two datasets are visualized in table 3.1 taken from [36].

Dataset	Released	Categories	total images	Images per Category			
				Min	Median	Mean	Max
Caltech101	2003	102	9144	31	59	90	800
Caltech256	2006	257	30607	80	100	109	827

Table 3.1: Comparison of Caltech datasets

It is to note that both Caltech datasets have a high within-class variability, meaning that objects are represented as photos, drawings or even cartoon-like. Some include background information, are cropped or rotated.

Columbia Object Image Library

The Columbia Object Image Library (COIL-100) [70] is an image dataset containing 7200 color images of 100 object categories. The images where generated placing the objects on a turntable and taking images using a fixed color camera at intervals of 5 degrees rotation. The images were size normalized and histogram stretched in a post-processing step. In contrast to the Caltech datasets, COIL-100 contain all the same black background, are cropped tightly and contain always the same physical object (although from different viewpoints). Illumination does also not change.

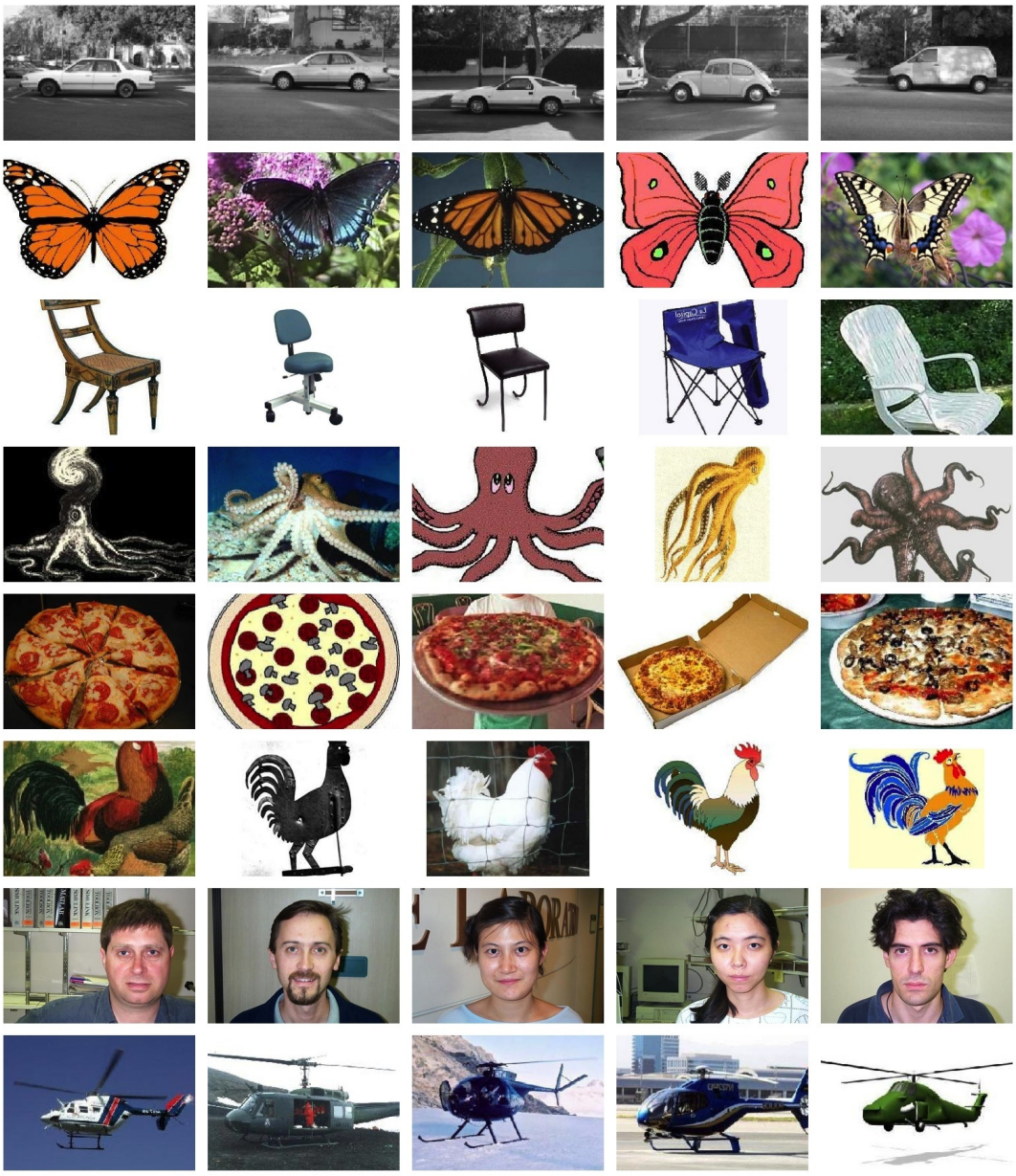

Figure 3.7: Caltech 101 Dataset
five samples from eight different object categories taken from Caltech 101 dataset: car_side,butterfly,chair,octopus,pizza,rooster,faces,helicopter.

ht]

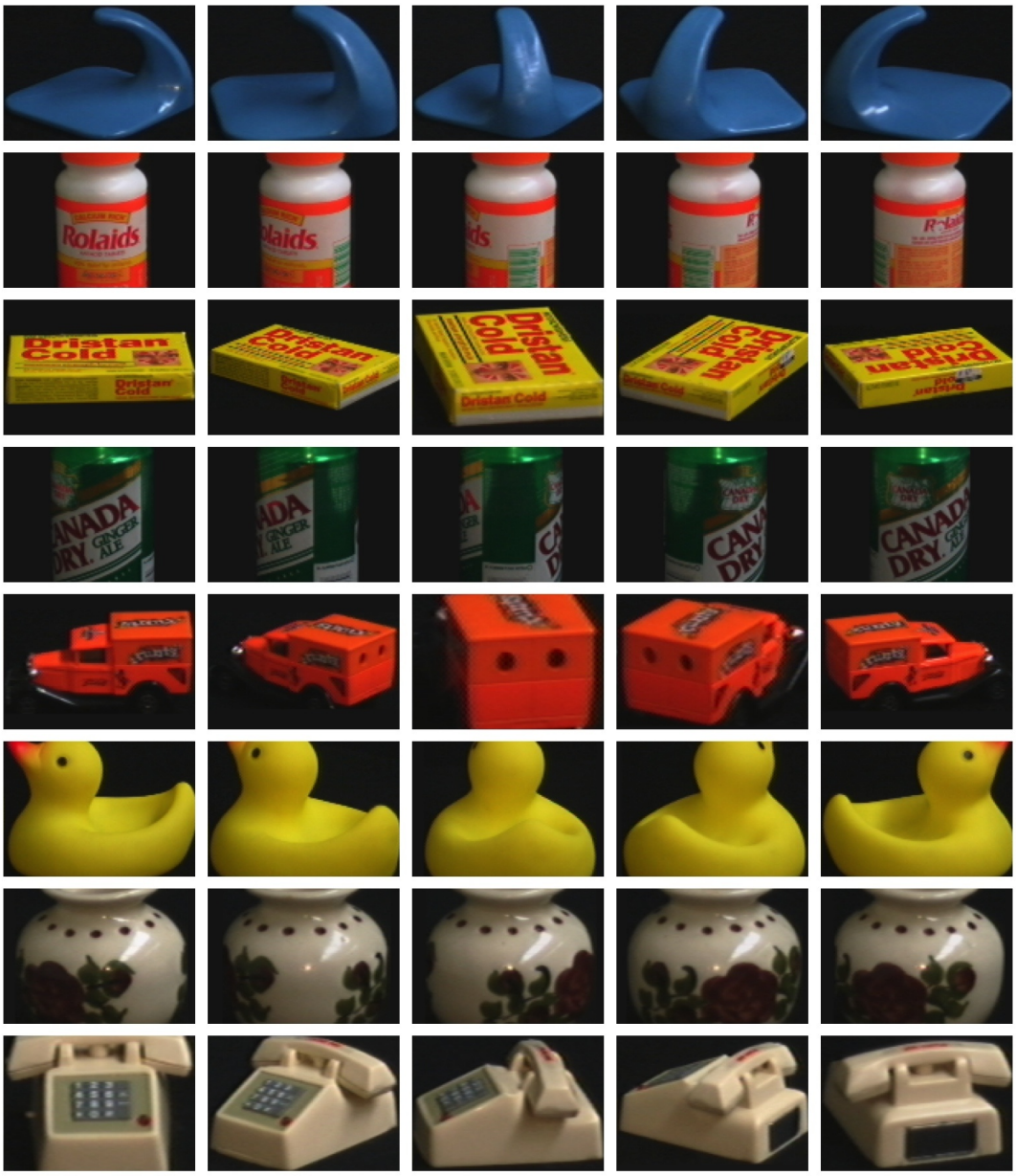

Figure 3.8: Columbia Object Image Library Dataset
five samples from eight different object categories taken from COIL-100 dataset:
Bail1,Bottle2,Box1,Can2,Car10,Duck,Vase2,Telephone.

3.6.2 Impact of Codebook Size

In the experiment described in this section we how the size of the visual codebook influences the classification for a supervised classification task on the Coil-100 dataset. A larger codebook implies that there are more basic elements the sparse coding algorithm can choose from to express the input image. Therefore we want to show that the classification accuracy improves for larger codebooks, however we hypothesize that from a certain codebook size onwards, increasing the size results in worse performance. This is expected mainly because at some point all dominant information can be incorporated into the codebook and adding more elements brings noise into the codebook which then influences the created descriptor and the final classification rate.

Experimental Setup

To conduct this experiment, eleven different k-dimensional codebooks for

$$k \in \{16, 32, 64, 128, 256, 512, 768, 1024, 1536, 2048, 4098\} \tag{3.65}$$

have been learned using the online codebook learning algorithm for sparse coding, described in algorithm 6. The codebooks have been created from the full Coil-100 dataset and have then be used to compute the final sparse feature descriptor using maximum pooling. Using this features a linear classifier based on the SVM algorithm has been trained for $i \in \{5, 10, 15, 20, 25, 30\}$ random samples per class. The samples not used for training have been used for evaluation. The achieved classification rates averaged over three trails are noted in table 3.4 where the best results are marked bold. A plot of the same data is available in figure 3.9.

Results

The results for the described experiment are contained in table 3.4: each row corresponds to the average classification accuracy for a specific codebook size and different number of training samples per class. The classification accuracy is weighted according to the class size to normalize the results with respect to the individual class sizes. The bold accuracy values represent the best result for a fixed number of training samples over all different codebook sizes.

The same information is represented as line chart in figure 3.9. The x-axis contains the different codebook sizes, the classification rate is represented on the y-axis and the colors code the different training sizes.

	30	25	20	15	10	5
16	0,942619	0,920638	0,916923	0,891930	0,850645	0,774030
32	0,947857	0,934894	0,926923	0,901228	0,866935	0,78
64	0,968333	0,961064	0,952885	0,932456	0,888065	0,805821
128	0,972143	0,964255	0,957308	0,931404	0,885968	0,791493
256	0,975000	0,967477	0,960192	0,941228	0,941228	0,805821
512	0,981667	0,967021	0,962692	0,938421	0,899516	0,800746
768	**0,984286**	**0,976383**	**0,959615**	**0,944561**	0,891774	0,811343
1024	0,977857	0,969362	0,959231	0,941754	**0,902097**	**0,811642**
1536	0,966667	0,952979	0,940385	0,914386	0,868387	0,759701
2048	0,968095	0,952766	0,943462	0,914912	0,869032	0,765970
4098	0,952143	0,941915	0,931538	0,906842	0,853226	0,760896

Table 3.2: Classification performance for different codebook sizes
average classification rate over 3 trails for the following codebook sizes
$size = \{16, 32, 64, 128, 256, 512, 768, 1024, 1536, 2048, 4098\}$ using $t = \{5, 10, 15, 20, 25, 30\}$
training samples per class. Highest classification rate per number of positive samples is
highlighted in bold.

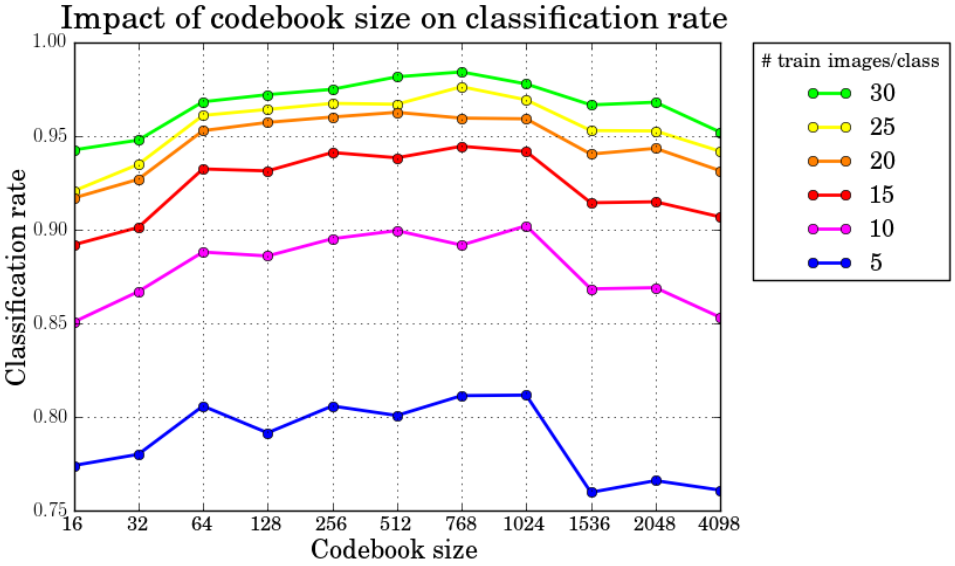

Figure 3.9: Classification performance for different sized codebooks for the Coil-100 dataset and $t = \{5, 10, 15, 20, 25, 30\}$ training samples per class. One can see that after a certain codebook size is reached (x-axis) classification performance for all numbers of training samples begins to drop.

Interpretation

The main outcome of this experiment show that the classification accuracy gets better for increasing codebook size, but, after reaching a certain size, it declines again. This indicates that, if the codebook is too small, many images from different classes could be roughly approximated by the same basis features, resulting in similar feature descriptors for different images. However on the other hand, if the codebook is to large, the classification accuracy declines again which could be the result of having very similar codebook elements. This phenomena is also visible in image 3.5 where a few basis elements are nearly the same.

It is however to note that even for very small codebooks (16,32 and 64 elements) the sparse features already lead to a good performance. The results show that codebook sizes from 512 to 2048 basis elements provide good results, which is also supported by recent publications like [109] or [96].

3.6.3 Impact of Codebook Quality

This experiment analyzes the impact of the codebook quality on the classification accuracy for a supervised classification task. With codebook quality we do not refer to the codebook size, whose impact has already been described previously in section 3.6.2 but rather to the information that is used to construct the codebook and therefore is used as basis elements during the coding stage. Therefore we created four different codebooks of equal size, where three of them are learned from the same dataset on which the classification task is performed, and one is generated from a totally different dataset.

Note that all codebooks have been learned using the same methodology, the only difference is the input data.

Our initial intentions before conducting this experiments were, that first of all that codebooks learned from few images perform worse than the ones trained on many images, simply because few images could not suffice to learn all important basis elements contained in the whole dataset. However we also argue that, if a codebook is created from a different dataset it could approach the quality and classification accuracy of a codebook trained on the original dataset given that the data covers a broad range of possible input.

Experimental Setup

As already mentioned, for this experiments we trained four different codebooks, each containing 1024 bases learned using Mairal's Online Dictionary Learning Algorithm described before in section 3.4. As basic feature descriptors, SIFT features extracted densely every 16 pixels have been used. As datasets, the Coil-100 and Caltech101 datasets have been used where the classification task has been performed on the Coil-100 dataset. Caltech101 has been used because first of all it does not contain any classes looking similar to the ones from Coil-100 and therefore codebooks trained on both datasets could contain different information. On the

other hand Caltech101 is also known to contain very diverse representations within the same class which range from photographs to hand drawings. Therefore Caltech101 is expected to contain a broad range of visual features.

Specifically the following codebooks have been created:

- *coil-100*: dataset created based on all images from the Coil-100 dataset
- $\frac{1}{2}$ *coil-100*: dataset created based on the first 50% images from the Coil-100 dataset
- $\frac{1}{4}$ *coil-100*: dataset created based on the first 25% images from the Coil-100 dataset
- *Caltech101*: dataset created based on all images from the Caltech101 dataset

Note that both codebooks, $\frac{1}{2}$ *coil-100* and $\frac{1}{4}$ *coil-100*, do not contain information from all classes, because of the way they are created.

Results

Table 3.3 contains the average classification accuracies over three trials for the four mentioned datasets. Figure 3.10 contains the same information plotted in a chart. The x-axis contains the number of training samples per class used, the y-axis represents the classification accuracy, and the different codebooks are color-coded.

dataset	5	10	15	20	25	30
coil-100	0.811642	0.891774	0.944561	0.959615	0.976383	0.984286
$\frac{1}{2}$ coil-100	0.673831	0.765161	0.809707	0.838397	0.862269	0.88
$\frac{1}{4}$ coil-100	0.580298	0.687258	0.739064	0.776153	0.797305	0.819286
Caltech101	**0.84602**	**0.921612**	**0.961462**	**0.975256**	**0.982553**	**0.988095**

Table 3.3: Classification performance for different codebooks
Average classification rate over 3 trails for the following 1024-dimensional codebooks: (1) codebook learned from the full coil-100 dataset, (2) codebook based on first half images from coil-100, (3) codebook based on first 25% of coil-100 and (4) a codebook generated from the whole Caltech101 dataset using $t = \{5, 10, 15, 20, 25, 30\}$ training samples per class. Highest classification rate is highlighted in bold.

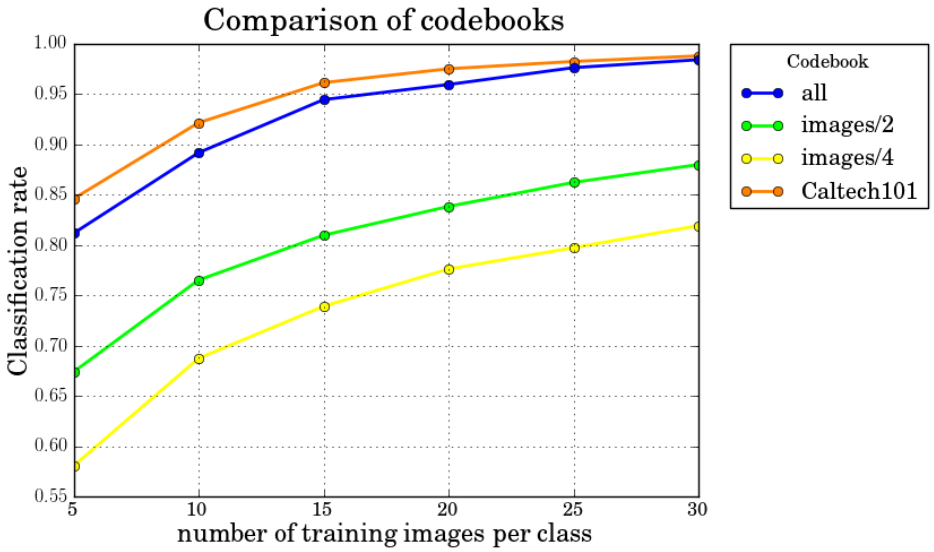

Figure 3.10: Impact of the codebook quality on classification performance for the Coil-100 dataset: all indicates that all samples from Coil-100 are used for codebook learning, images/2 and images/4 indicate that $\frac{1}{2}(\frac{1}{4})$ of the dataset images have been randomly chosen to generate the codebook, Caltech101 is a codebook generated from a different dataset (the Caltech101 dataset)

Interpretation

From both representations of the experimental data it is clearly visible that the feature descriptors based on the Caltech101 codebook provide best classification results. It shows also that the difference to the second-best performing codebook (based on the full Coil-100 dataset) diminishes for increasing numbers of used training samples. The codebooks trained on a subset of the Coil-100 dataset provide the worst performance, where the codebook trained on the fewest training data performs worst.

The first observable behavior that this data shows that the final classification is better for codebooks learned from more data. This seems to be intuitive since when learning the codebook on many data samples the important basis can be better represented than by only sampling a subset of the available features. The more surprising fact is however that a codebook learned from a totally different dataset (compare figures 3.7 and 3.8) can achieve and even outperform the codebook learned on the full testdataset. The Caltech101 codebook, based on the diverse visual informations even not present in the Coil100, could provide a good basis to express the Coil100 features, even though it might contain basis components that are even not present in the Coil100 dataset. This whole finding implies that for certain application scenarios it is not even needed to create and learn a specific codebook. It is enough to have available one trained codebook that contains a broad range of the feature

domain, possibly also created with a synthetic dataset. This codebook could then be reused in many applications without the need to use algorithms like Mairals Online Codebook Learning to create a new codebook every time.

It is however to note that in the previous comment we combined two different viewpoints. The one that is supported by our data shows that, as already mentioned, given enough data to learn the codebook, it can outperform even domain-specific ones. The Online Codebook Learning algorithm on the other hand aims to extract the relevant basis in an iterative fashion. Such a codebook therefore just contains the relevant information needed to express the given data, it could therefore be smaller than a 'general' codebook. It can also contain more fine-grained information, useful to express the specific data, but that would be irrelevant when creating a general codebook.

Therefore the codebook learning algorithms can therefore be used to either create very data-specific or more general codebooks, depending on the possible usage scenarios. These findings are also supported by a recent publication of Coates et al [20], studying the importance of the encoding for Sparse Coding and Vector Quantization.

3.6.4 Pooling Strategies

If we consider U as the solution of the sparse coding equation 3.58 on the set of local descriptors $X = x_1, x_2, \ldots, x_M$, using a fixed codebook D we can define different pooling functions to compute the final feature vector. Since U contains the responses of each local feature descriptor to the items in the codebook D, different pooling functions construct different feature vectors. In this section we define several different pooling functions and analyze their impact on the final classification rate similar to experiments done in [102, 96, 14].

The general pooling function can be defined like in [102] as $z = F(U)$ where F is defined on each column of U, since each column contains the responses of each local image feature to one particular dictionary entry D_k. For our experiments we defined the following pooling functions:

maximum pooling:

$$z_j = max\{u_{1j}, u_{2j}, \ldots, u_{Mj}\}$$

min pooling:

$$z_j = min\{u_{1j}, u_{2j}, \ldots, u_{Mj}\}$$

average pooling:

$$z_j = \frac{1}{M} \sum_{i=1}^{M} u_{ij}$$

random pooling:

$$z_j = u_{ij} \text{ where } i = \text{ rand } (1, \ldots, M)$$

absolute maximum pooling:

$$z_j = max\{|u_{1j}|, |u_{2j}|, \ldots, |u_{Mj}|\}$$

mean squared square root pooling:

$$z_j = \sqrt{\frac{1}{M} \sum_{i=1}^{M} u_{ij}^2}$$

mean absolute pooling:

$$z_j = \frac{1}{M} \sum_{i=1}^{M} |u_{ij}|$$

where $|x|$ is the absolute value of x, z_j denotes the j-th entry in the final feature vector and u_{ij} is the element of U in the i-th row and j-th column.

Experimental Setup

The following steps describe the basic experimental procedure and setup to evaluate the different feature pooling strategies. For our experimental evaluation of the different pooling strategies we followed the Locality-constrained Linear Coding method proposed in [96] using Caltech 101 [26] as dataset.

- **Feature extraction:** As basic feature descriptor we used the Scale Invariant Feature Transform (SIFT), densely extracted from a 6 pixel grid, using a patch size of 16 pixels. Using this method for every image within the dataset we extracted between 850 and 1400 SIFT features depending on the input image dimensions.
- **Codebook creation:** The experiment relies on a pretrained codebook (denoted by D in 3.58) created using the K-Means clustering to extract the 1024 basis features from the whole dataset.
- **Pooling:** After the projection of the image features onto the codebook we used the different pooling strategies to extract a descriptor for the image features.
- **Classification:** For the classification task we used a linear multiclass Support Vector Machine trained on $\{5, 10, 15, 20, 25, 30\}$ images and evaluated on the remaining ones. The reported classification rate is a weighted average over the individual classes to take into account that some classes contain more items than others.

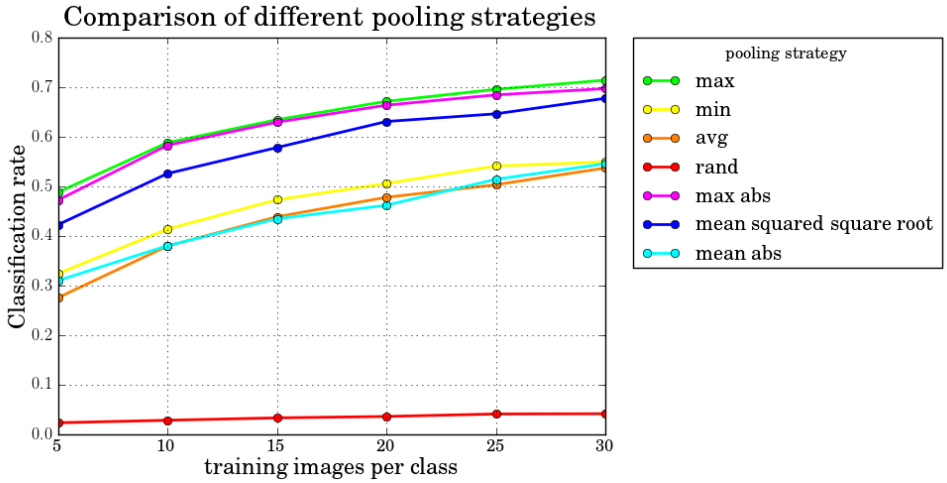

Figure 3.11: Comparison of the classification accuracy for different spatial pooling strategies

Results

The results of the described experiment are presented in table 3.4. The best classification performance for different numbers of training images per class are highlighted in bold an indicate clearly that maximum pooling is the best-performing feature pooling strategy. The same information is also presented in plot 3.11, where the relation between the different pooling strategies is easier visible. All mentioned results were averaged over three trials in order to minimize the influence of the randomly chosen training samples per class.

training images per class	(1) max	(2) min	(3) avg	(4) rand	(5) max abs	(6) mean root	(7) mean abs
5	**0,488**	0,324	0,275	0,023	0,472	0,422	0,310
10	**0,586**	0,413	0,379	0,028	0,582	0,525	0,380
15	**0,634**	0,473	0,438	0,033	0,629	0,578	0,434
20	**0,671**	0,505	0,477	0,036	0,663	0,630	0,461
25	**0,695**	0,541	0,503	0,041	0,684	0,646	0,514
30	**0,714**	0,549	0,537	0,041	0,697	0,677	0,545

Table 3.4: Classification performance on different feature pooling strategies: average classification rate over 3 trails for the following pooling strategies: (1) maximum pooling, (2) minimum pooling, (3) average pooling, (4) random pooling, (5) absolute maximum pooling, (6) mean squared square root pooling, (7) mean absolute pooling. Best result is highlighted in bold

Interpretation

The data shows that there is almost no difference between maximum pooling and absolute maximum pooling. The same holds for average and mean absolute pooling. This can be explained by the fact that non-negative sparse coding is used, leading to no negative activations and therefore to no difference between absolute and non-absolute measures.Recent publications [83, 30] indicate that maximum pooling is biologically plausible, meaning that with a high probability the from single neuron activations in mammal brains also the ones with highest activations are used. This implies that the environment we observe is the maximal response of external inputs arriving in our visual cortex. Boureau et al performed also an analysis of feature pooling [15] in recognition systems and confirm the results of this experiment including the importance of maximum pooling.

4 Applications

This chapter contains two exemplary use cases for the sparse feature descriptors discussed in chapter 3.

In the first application, the sparse feature descriptors are used to predict class changes in real-world unlabeled video segments by exploiting slowly changing uncorrelated features available in the feature descriptor. Experiments on two different datasets highlight the performance of the proposed method.

The second application in chapter 4.2 deals with the task of unusual event detection by analyzing and evaluating the performance of different motion- and appearance-based feature descriptors.

These tasks have been chosen to evaluate the performance of the Sparse Coding feature representation, as both tasks need a robust and compact description of the video sequences. For the task of scene change detection, it is desired to have a compact description while preserving all information needed to detect the scene changes as they appear. In contrast, for the task of unusual event detection the descriptor has to bee robust to appearance changes occurring within the normal events. People walking down a street change their appearance because of their body movement, different perspective and possible occlusions. The feature descriptor, however should be capable of finding a common representation of those variances but at the same time effectively discriminate them from abnormal events.

4.1 Class Change Detection in Unlabeled Video Sequences

This section describes an application scenario for the feature descriptor based on sparse coding where it is used to analyze unlabeled video sequences. After stating the problem we want to solve in section 4.1.1, some experiments in section 4.1.4 are explained and the corresponding results (section 4.1.5) we can conclude that sparse coding is a powerful tool for sequence-based data analysis.

The presented system [86] analyzes the feature descriptor change over time using slow feature analysis (SFA) and interprets the most dominant signal variation as visual class changes. It therefore provides a novel, unsupervised approach to automated labeling of image sequences.

Figure 4.1: The temporal coherence principle: with high probability two consecutive frames contain the same object(s), even though characteristics like lighting, viewpoint or scale might change

4.1.1 Problem Definition

The problem to be tackled by the application is the following: Given a video sequence, find the most dominant changes present in the video. This changes can then be interpreted in a classification sense as class changes, as they are expected to change the underlying feature descriptor at most. Note that the input video data is unlabeled, meaning that there is absolutely no information present about any video frame except the pixel values itself. For the video sequence depicted in figure 4.1, the detected classes would therefore not be *elephant* and *gazelle*, but rather *class1* and *class2*, as the algorithm has no notion of class names and should only detect class changes.

These labels are useful, and in fact needed, when training classifiers for object recognition tasks. Such classifiers are normally trained on positive and negative samples for all possible object categories and should include a wide range appearance variations. This leads to a huge amount of needed training data, and hand labeling quickly becomes unfeasible. Negative samples are usually easier to generate: starting with a few training images a classifier is trained and evaluated on images not containing the object. The wrongly classified images are then used as negative samples to retrain the classifier. Automatically generating positive labels for different object categories is harder. Some applications rely on a correctly initialized and learned tracker, following the object class of interest and therefore generating positive labels. All of the mentioned methods, however, rely on some sort of initial setup or information. In the following we propose an unsupervised method for class label generation, relying solely on the video sequence and its incorporated information.

To solve such a task there are various possible solutions, but all include tracking the behavior of a feature descriptor over time. Under the assumption that the different classes present in the video are differently colored, one could simply track the pixel intensities as a histogram over time. If the color change from one frame to another (or within a pre-defined window size) surpasses a given threshold a class change is detected.

However, this choice of feature descriptor has some limitations. First of all it is not invariant to varying light conditions as they might occur within a video sequence. If at some time the light increases, the histogram will reflect the chances and might detect a new class while

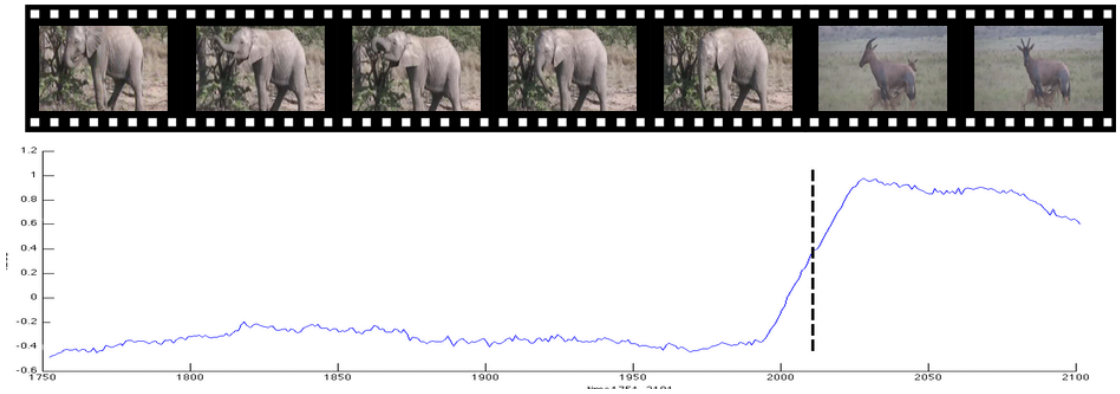

Figure 4.2: Illustration of a video sequence with the corresponding first slow feature indicating a rapid change

the same object is present in the video only under other light conditions. The next problem is that the assumption that different classes would be composed of different colors might be totally wrong. Thinking of a video sequence containing an elephant in the wilderness, followed by a hippopotamus on land, both color histograms would look nearly the same. They would mainly contain gray values and could lack red and blue. Other possible problems include partial occlusions or heavy spatial transformations due to zooming and shaking with a camcorder, resulting in a drastically changed feature descriptor.

On the other hand the described sparse feature descriptor already has the property that it changes drastically from one to another video frame even if they share most visual information. This can be seen in the illustration 4.3 where five feature descriptors for consecutive video frames are plotted.

The vertical bars indicate a positive activation, all other entries in the feature descriptor are zero. The color indicates the activation intensity where red is the maximum intensity and blue the minimal one. One might suspect that for nearly equal input the activations for the linear combination on the same codebook are nearly the same or show at least some common basis activations. However as seen in the figure this is not the case. Note that the depicted features are created based on SIFT features on a rather large codebook and that the factors like codebook size, spatial pooling function and basis features impact these results.

The underlying assumption for such a task to work is that the frequently changing sparse codes, although looking diverse, carry some common information. One method that is capable to identify individual slowly changing information from a single input source is the Slow Feature Analysis (SFA) which is described in detail in the following section 4.1.2.

The entire process of unsupervised labeling the classes in a video sequence is summarized also in the overview plot of figure 4.4. It shows that first of all the features of the video

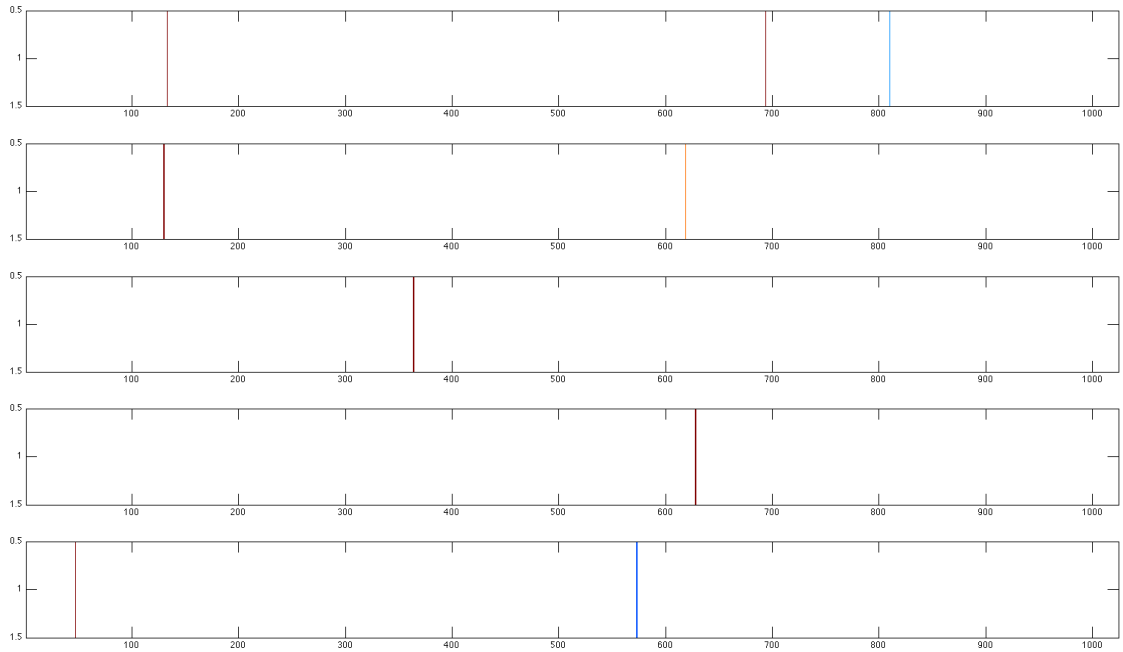

Figure 4.3: Five sparse feature descriptors calculated from consecutive video frames, where the color is mapped to the activation intensity, all other activations are zero. It shows that although the consecutive video frames contain nearly the same information the sparse feature descriptors are quiet different.

frames have to be extracted, as mentioned previously in section 2, and these features are then used as input for the slow features analysis algorithm (described in section 4.1.2). After the analysis, the decision if the video sequence contains class changes is made based on the slowly varying signals. If the signals change by more than a certain pre-specified threshold, than the algorithm decides that this indicates a class change, which is then labeled. In this way, this processing queue should be able to detect the changes between different objects or scenes that are shown in the video: In the figure the change between an elephant (in the first part of the video sequence) and a gazelle (the last part of the shown sequence) is detected.

Available scene change or cut detection algorithms for video sequences mostly apply statistical principles to evaluate feature descriptors [57, 84]. Another reproach is followed in [42] where segmentation and tracking methods are combined to detect scene cuts. Additionaly there exist also methods using motion features [71] and [105] raw pixel values to accomplish the mentioned task.

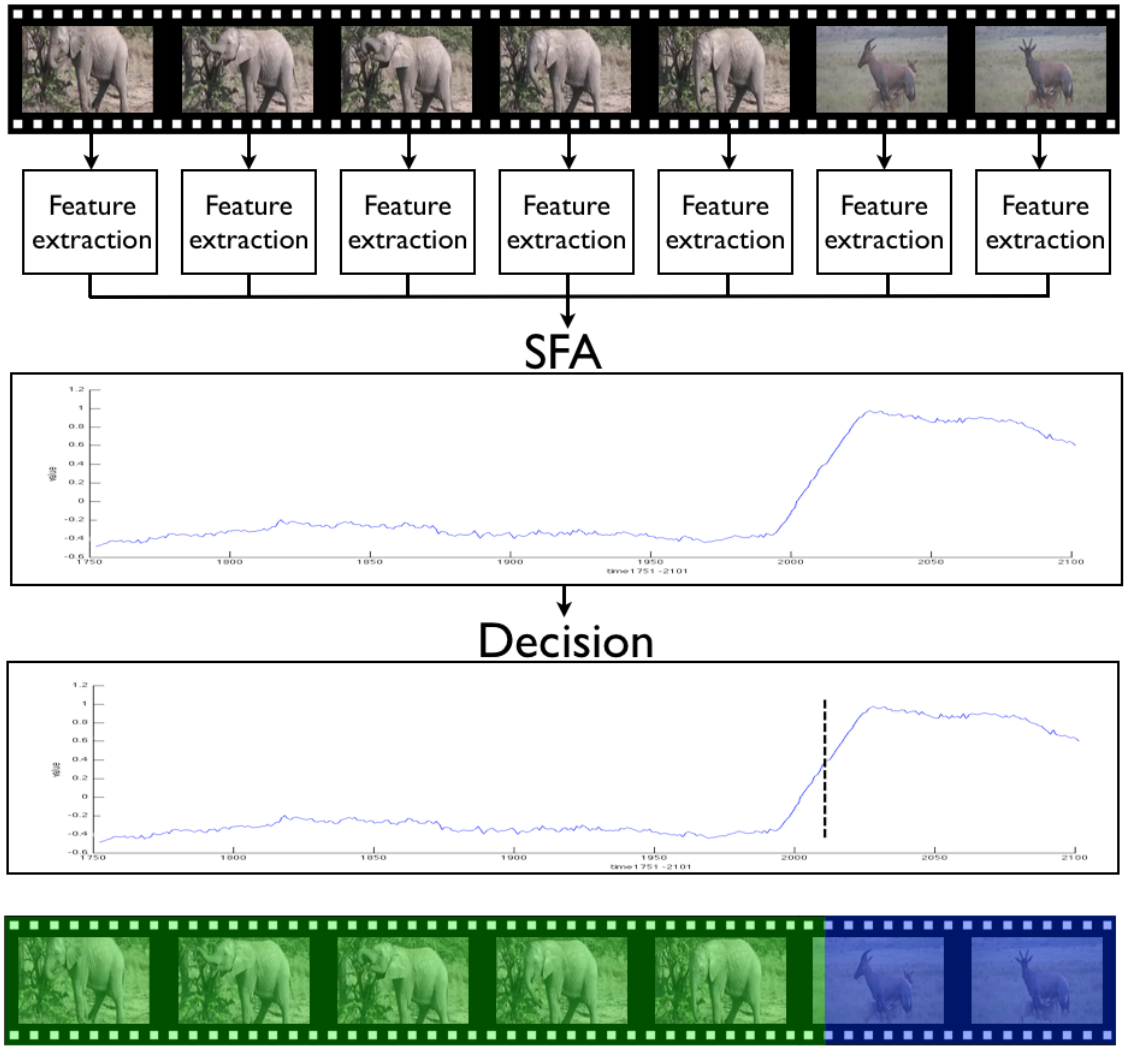

Figure 4.4: Proposed pipeline for frame labeling based on Slow Feature Analysis: after the extraction of a basic feature descriptor for each individual frame, batch-based SFA is computed. Using a threshold function label changes are detected and the frames are labeled accordingly.

4.1.2 Slow Feature Analysis

Slow Feature Analysis (SFA) describes an unsupervised method to learn slowly varying or invariant signals from a single input signal. Its power is based on the fact that it can extract a high number of decorrelated features from the input signal, ordered by their slowness, and that it is guaranteed to find the optimal solution within a set of linear expansions. Formally the problem can be formulated as in [98]:

Given an input signal $x(t) = [x_1(t), x_2(t), \ldots, x_I(t)]$ with $t \in [t_0, t_1]$, the goal is to find an input-output mapping function $g(x) = [g_1(x), \ldots, g_J(x)]$ generating the output signal $y(t) = [y_1(t), \ldots, y_J(t)]$ with $y_j(t) = g_j(x(t))$ such that for each $j \in 1, \ldots, J$

$$\Delta_j := \Delta(y_j) := \langle y_j^2 \rangle \text{ is minimal} \tag{4.1}$$

under the following constraints:

$$\langle y_j \rangle = 0 \qquad \text{(zero mean)} \tag{4.2}$$

$$\langle y_j^2 \rangle = 1 \qquad \text{(unit variance)} \tag{4.3}$$

$$\forall j' < j : \langle y_{j'} y_j \rangle = 0 \qquad \text{(decorrelation)} \tag{4.4}$$

, where $\langle \quad \rangle$ expresses temporal averaging. The variation of the temporal output signal is minimized by Eq.(4.2), constraints in Eq.(4.2) and Eq.(4.3) avoid the trivial constant solutions $y(t) = const$ and constraint Eq.(4.4) ensures that the output signals do not reproduce each other but contain different information.

This is generally a hard problem of variational calculus, meaning that instead of functions, functionals are extremized, but if g_j are constrained to a fixed set of linear combinations of nonlinear functions, the problem becomes feasible.

Learning the slow features of an input signal consists of the following steps:

- **Input:** The input to the algorithm is the I-dimensional vector $\tilde{x}(t)$.
- **Normalization:** $\tilde{x}$ is normalized to obtain the normalized input signal $x(t) = [x_1(t), \ldots, x_I(t)]$ to satisfy Eq.(4.2) and Eq.(4.3).
- **Nonlinear expansion** An arbitrary nonlinear expansion function $\tilde{h}(x)$ is used to construct the signal $\tilde{z}(t)$. Common choices for $\tilde{h}(x)$ are all monomials of degree one (linear SFA) and degree two (quadratic SFA).
- **Whitening:** $\tilde{z}(t)$ is again normalized to satisfy the zero mean and unit variance properties. This step is often called whitening or sphering phase.
- **Principal Component Analysis:** PCA is performed on the temporal normalized signal $\tilde{z}(t)$, the resulting eigenvectors with the smallest eigenvalues are used as weights to compute the slowly varying functions.
- **Application:** The slowest n SFA signals can then be applied to new test data.

A simple toy example from [98] is used to illustrate the involved steps and the outcome in more detail. Suppose there exists an input signal $x(t)$ in the two-dimensional input space which is defined as the combination of two individual signals $x(t) = x_1(t) + x_2(t)$ for each time step t. The two individual signals are defined as $x_1(t) = sin(t) + cos(11t)^2$ and $x_2(t) = cos(11t)$ which are two rapidly oscillating functions. However they share a common 'slow' signal $y_{slow}(t) = sin(t)$ which can be shown with the following simple equation:

$$\begin{aligned} y(t) &= x_1(t) - x_2(t)^2 \\ &= (sin(t) + cos(11t)^2) - (cos(11t))^2 \\ &= sin(t) \end{aligned} \tag{4.5}$$

. We are now interested in this slow varying function $g(x)$ which we have seen is $g(x) = x_1 - x_2^2$. The first step in the SFA algorithm performs the transformation of the non-linear problem into a linear problem by expansion into a predefined subspace. As we know that our desired slow function is a polynomial of degree two, we use this subspace for our example. Therefore our new signals in the expanded space are $z_1 = x_1, z_2 = x_2, z_3 = x_1x_2, z_4 = x_1^2$ and $z_5 = x_2^2$. Since every polynomial of degree two can now be formulated as a linear combination of these five signals, our problem becomes linear. The next step involves the normalization of the expanded signals to satisfy the constraints in Eq.(4.2), Eq.(4.3) and Eq.(4.4). After the constraints are satisfied in the normalized space, the time derivative $\dot{z}(t)$ is calculated which indicates the temporal variation. The signal with the lowest variance in the time derivative $\dot{z}(t)$ represents now the slowest varying signal $g(x)$ in the normalized feature space. The slowest signal in the input space is then computed as $y(t) = g(x(t))$.

To show that the concept of slowly varying functions can be exploited in numerous domains, some recent publications are shortly summerized.

Recently it has been shown in [44] that the unsupervised Slow Feature Analysis algorithm can approximate the classification capability of the supervised Fisher's Linear Discriminant Analysis under the assumption that temporal coherence is present in the input data, meaning that temporally near samples are likely to be from the same class.

In [7] the Slow Feature Analysis algorithm was used to perform supervised pattern recognition on the MNIST dataset. The author showed that, provided that the classes are non-overlaping and the function space is large enough, the slowly varying functions learned by SFA respond similarly for input vectors belonging to the same class.

Another application of SFA has recently been published in [49] where the slow signals together with color information and textural features are used as appearance descriptors to train a patch-based classifier for a street segmentation task using a single camera.

In [45] the first online variant of the Slow Feature Analysis algorithm has been published. The authors derive the online algorithm using a combination of incremental Principal Component Analysis and Minor Component Analysis and evaluate its performance in various experiments. In [72] SFA is performed on image patches extracted from an autonomous mobile robot

camera to retrieve visual features. Using a hierarchical network of SFA layers, in [28] object recognition is performed. The results indicate that SFA can learn robust representations for slowly occuring transformations.

On a more theoretical level in [93] the authors give a maximum likelihood interpretation of the SFA algorithm and show that their probabilistic model can successfully deal with noise. Konen and Koch analyze in [47] the slowness concept in SFA and come to the conclusion that this slowness depends on the driving force of the underlying time series components. In [9] Blaschke et al could show that linear Slow Feature Analysis and second order Independent Component Analysis are formally equivalent, implying that statistical independence and slowness have the same expressability in this case.

4.1.3 Implementation

The involved components and steps are described based on the illustrative pipeline in figure 4.4 which can be divided into the following main steps: basic feature extraction, slow feature analysis and class segmentation or decision.

Basic Feature Extraction

As a first step from the input video/images a basic feature descriptor is computed. Given a set of input images $\{i_1, i_2, \ldots, i_n\}$, for each one a feature vector f is computed. For evaluation and comparison purposes the following feature descriptors have been extracted:

- **SC**: the sparse feature descriptors discussed in chapter 3. The descriptor is computed by using SIFT features extracted densely on a 6×6 pixel grid, a 1024-dimensional codebook calculated based on the Caltech101 dataset and the maximum pooling operation.
- **BOF**: a normalized bag-of features descriptor based on SIFT features extracted on key points.
- **PHOG**: Pyramid Histogram of Oriented Gradients descriptor, a HOG descriptor computed from a scale pyramid, Used 80 histogram bins and an angle of 360 degrees.

The most interesting comparison, without any prior knowledge, might be the bag-of-features descriptor as it is also (usually) a sparse descriptor although its sparsity is not enforced by any constraint or even guaranteed. If an image contains all features present in the codebook, which could happen if the codebook size is too small, then the descriptor is not sparse anymore but has low activations on every dimension.

Slow Feature Analysis

Based on the set of basic feature descriptors $\{f_i, \ldots, f_n\}$ the k slowly varying functions are extracted using the MATLAB toolkit sfa-tk[6] in an extended version that improved the numerical stability [46]. The original implementation runs into numerical instabilities if the covariance matrix in the expanded signal space does not have full rank. To overcome this problem, the modified version uses singular value decomposition and adds small noise to force full rank. Even though an incremental version of the SFA algorithm has been proposed in [45], the current implementation performs batch-based slow feature analysis. For each such patch of feature descriptors a fixed number of slowly varying functions in the function space of all polynomials of degree 2 are calculated performing the steps mentioned in section 4.1.2. Since the amount of data to be processed is quite small because the SFA is not performed on the raw pixel data but on the compressed information available in the basis feature descriptor we would not benefit much from the incremental approach in our setting.

Class Segmentation

Based on the computed slow signals it is relatively easy to detect the biggest changes in the signal. As the information needed to discriminate N classes is contained in the first $N-1$ slow functions (as shown in [28]), this problem reduces to finding the biggest changes in the desired amount of functions. After first experiments with a threshold-based approach, based on the normalized signal value, its window-based mean and standard deviation values, the approach that has found to work best has been to first slightly smooth the slow signals and then recursively calculate the maximal difference between adjacent signal values. Signal smoothing can be easily performed using the MATLAB *smooth()* functions contained in the Curve Fitting Toolbox. Calculating the maximal recursive difference in MATLAB can be performed using the function call `[value index]=max(diff(diff(diff(signal))));` or `[value index]=max(diff(signal,3));` in short notation where `signal` contains a single slow feature and `index` will contain the index with the maximal change in one signal. Since it could happen that within such a sliding window there exist multiple or even no class changes we assume as a precondition that for every window the exact number of class changes is known in advance.

4.1.4 Experiments

To evaluate if the feature descriptor based on sparse coding contains enough information to discriminate the classes after the slow feature analysis, we performed experiments based on two datasets. The first dataset is a 3:30 minutes long amateur video called *Baby Wild Animals!* retrieved from the video portal Youtube[1] and will later on be referenced as *animal dataset*. As a first step a ground truth dataset for the video has been created by manually marking the changes from one animal to another which are summarized in table 4.1. Note that these frame numbers are not necessarily 100% correct as it is sometimes really hard

[1] `http://www.youtube.com/watch?v=nDDDqF1rpjM`

to decide at which exact moment one animal disappears and a new comes in, especially in the case of second-long cross-fading or zooming. Note that the whole application relies only on unlabeled data and unsupervised methods therefore the labels (animal names) are only needed for evaluation purposes not for the SFA algorithm itself.

frame	animal
1-20	Intro
21-375	Elephant
376-615	Gazelle
616-775	Monkey
776-1200	Monkey 2
1201-1530	Rhinoceros
1531-1840	Zebra
1841-2230	Lioness
2231-2450	Lioness 2
2451-2830	Gazelle 2
2831-3220	Hyenas
3221-3780	Zebra 2
3781-4080	Rhinoceros 2
4081-4310	Rhinoceros 3
4311-4700	Outro

Table 4.1: The fifteen manually identified classes and their appearance in the video indicated by the frame numbers.

The second dataset contains short video sequences from people crossing a road and will be further on refered to as *extended tracks dataset*. It consists of a total of 551 frames belonging to 13 people (classes) and has been used in [3]. The class with the most samples contains 80 frames whereas the smallest class only contains 7 video frames. The exact frames per class are listed in table 4.2

The aim of the first conducted experiment was to analyze how well the proposed architecture works in predicting the class change based on the sparse descriptor and the change in the slow functions. Therefore for both datasets the steps described above and illustrated in the pipeline 4.4 were performed. At the end of the pipeline the algorithm predicts the exact frame where the class change happened and is then compared to the labeled ground truth. These results are presented and discussed in section 4.1.5.

The next experiment had a different goal. Having the predicted class changes from the previous experiment and knowing that some of them might be labeled wrong, we tried to analyze the consequences of the mislabeling. The main idea is the following: having an unlabeled video sequence it could be possible to predict the labels using SFA and based on

frame	class
1-30	1
31-70	2
71-13	3
140-161	4
162-210	5
211-233	6
234 -279	7
280 - 326	8
327-370	9
371-417	10
418-496	11
497-529	12
530-551	13

Table 4.2: The frames and corresponding classes for the extended tracks datasett

that labels perform supervised classification. In short this means that our architecture could perform classical supervised classification by creating labels using an unsupervised learning method based on extracting slowly varying signals. Based on the estimated class labels we trained a standard linear SVM and evaluated it on the ground truth labels. For comparison we also trained a model based on the correct labels. The results in section 4.1.5 indicate that even though some labels are wrong and therefore the SVM learns on incorrect data its classification performance does not drop by a significant amount.

In addition to evaluating the classification performance using the estimated class labels, we also evaluate the classification performance on newly, unseen objects. Our proposed method could be feed with hours of different video material, and if the video material covers most of the possible appearance variations within an object class, even objects not coming from the input video sequences should be classified correctly. Snapshots of the training video sequences for classes *elephant* and *zebra* are represented in figure 4.5. Samples of correctly and wrongly classified samples, retrieved using Google Image search, are highlighted in green and red. The classification accuracy on this unseen objects is highly dependent on the information available within the video sequence, therefore no accuracy measure is reported for this experiment. The image, however, clearly shows that poses contain in the training sequences could be correctly classified (e.g, zebras from side view), while most misclassified samples contain poses not present in the training data (e.g., elephant from the back or lying zebras).

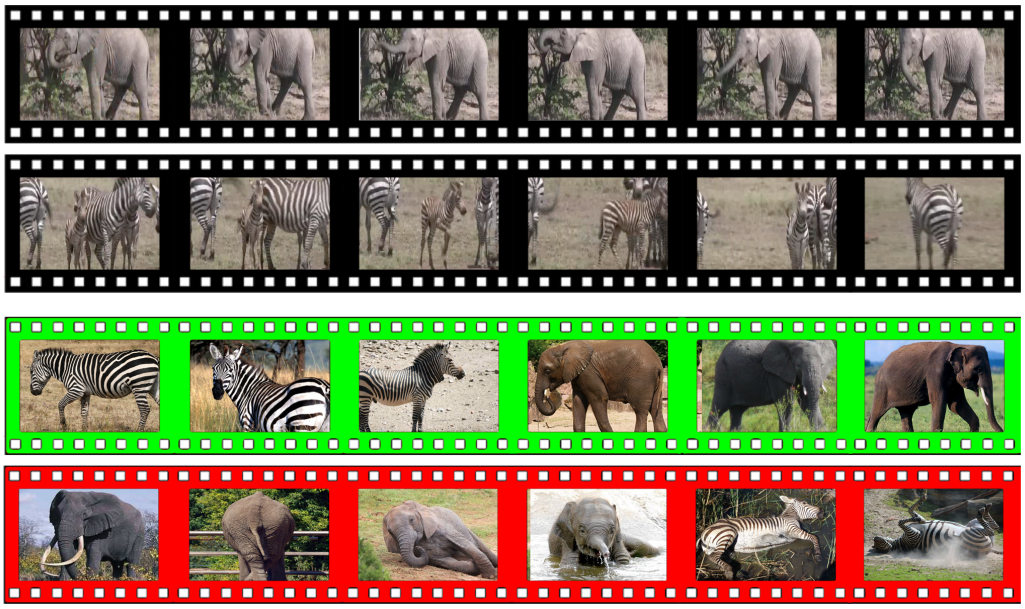

Figure 4.5: Samples of correct and wrong classified animals for categories elephant and zebra. The top two filmstrips illustrate the video sequences used to learn the classifier, the green and red filmstrips contain correctly and wrong classified animals, respectively.

4.1.5 Results

This chapter gives the results of the conducted experiments, which analyze whether slowness can be used to effectively discriminate visual object categories.

The first result is given in table 4.3, which contains a comparison of the detected versus the actual class borders for the animal dataset. The numbers indicate the frames. within the video.

class	detected	ground truth	missdetected frames
Intro	1-23	1-20	3
Elephant	24-382	21-375	10
Gazelle	383-616	376-615	8
Monkey	617-928	616-775	154
Monkey 2	929-1072	776-1200	281
Rhinoceros	1073-1530	1201-1530	128
Zebra	1531-1844	1531-1840	4
Lioness	1845-2236	1841-2230	10
Lioness 2	2237-2458	2231-2450	14
Gazelle 2	2459-2893	2451-2830	71
Hyenas	2894-3328	2831-3220	71
Zebra 2	3329-3642	3221-3780	146
Rhinoceros 2	3643-3993	3781-4080	225
Rhinoceros 3	3994-4317	4081-4310	94
Outro	4318-4700	4311-4700	9

Table 4.3: Class borders detected by the SFA algorithm and the manual annotated ground truth for the animal dataset

The next table 4.4 contains the results for the same experiment based on the extended tracks dataset.

For both datasets the classification capabilities based on the probably wrong detected class borders has been evaluated and the results are visible in tables 4.5 and 4.6. For all test we used a changing number of samples for training the SVM and testing the learned model by taking every x-th training sample and evaluating on the rest. The exact numbers of training and test samples are visible in the first two columns. The result of training the SVM on the class labels extracted from the SFA signals and tested on the ground truth labels is contained in columns 3 and 4 whereas the remaining columns contain the results based on training and testing on the ground truth data. The accuracy is expressed as mean over the individual class accuracies, weighted by their class-size to prevent that misclassification in small classes have higher influence. The indicated standard deviation is the standard deviation of all individual accuracies. Therefore a low standard deviation indicates that all individual classes can be

class	detected	ground truth	missdetected frames
1	1-30	1-30	0
2	31-70	31-71	1
3	71-139	72-139	1
4	140-161	140-160	1
5	162-210	161-209	2
6	211-233	210-216	8
7	234 -279	217-272	24
8	280 - 326	273-326	7
9	327-370	327-370	0
10	371-417	371-416	1
11	418-496	417-496	1
12	497-529	497-529	0
13	530-551	530-551	0

Table 4.4: Class borders detected by the SFA algorithm and the manual annotated ground truth for the extended tracks dataset

recognized equally good while a high deviation is a sign for having some classes that can be recognized better than others.

Another experimental goal was to qualitatively compare the slow signals extracted by the SFA algorithm for different feature descriptors. Since the output SFA values depend on the input signals, which have different value ranges it is not possible to compute some distance measure on these signals for the different features.

Therefore for a quick analysis the first slowly varying signal for the three already mentioned feature descriptors and for different frames are illustrated in figure 4.6.

4.1.6 Discussion

From the experimental results showed in the previous section, it is clearly visible that the slow components extracted using SFA serve as a good indicator for visual class changes. Given that the video is encoded with 30 frames per second, even the class with the most miss detected frames (Monkey 2) fails just for a few seconds in the original video. Given that between some classes the video contains long fade-in, where no or even both animals are visible this seems to be a good result. It is also to note that the manual labeled ground truth frames are not necessarily 100% correct as sometimes it is even difficult for humans to accurately annotate the moment when one class ends and a new one starts. It is to note that generally the number of miss detected frames for the extended tracks dataset is much lower than the one for the animals dataset. One reason for this might be contained in the dataset itself. The animal dataset is a video created for a human audience, not a synthetic dataset. It therefore contains blurred frames, jittery video sequences and animated scene changes.

# samples		train:sfa - test:ground		train:ground - test:ground	
train	test	mean accuracy	standard deviation	mean accuracy	standard deviation
2347	2353	0.9946	0.00460016	0.992344	0.00485368
1562	3138	0.994408	0.00592073	0.990439	0.00718787
1168	3532	0.992399	0.00671143	0.98932	0.00837344
932	3768	0.991111	0.00768069	0.988714	0.00774758
581	4119	0.989001	0.0098038	0.987507	0.00961077
464	4236	0.987952	0.00846541	0.987232	0.0100202
305	4395	0.98613	0.0127703	0.983767	0.0137247
227	4473	0.963857	0.0550626	0.971422	0.0426032
168	4532	0.887729	0.264721	0.892406	0.250044
127	4573	0.882995	0.258925	0.876673	0.259023
87	4613	0.856727	0.261359	0.874702	0.247859
56	4644	0.804432	0.306633	0.794059	0.326191
41	4659	0.78056	0.259704	0.78537	0.24276
25	4675	0.660694	0.395944	0.664635	0.327906
16	4686	0.535826	0.327945	0.624041	0.319573

Table 4.5: Classification results using a linear Support Vector Machine for class labels predicted by SFA and the manually annotated ground truth labels for the animals dataset. Even tough the SVM learned on a few wrong labels (see table 4.3) the classification accuracy is comparable to the ground truth accuracy.

# samples		train:sfa - test:ground		train:ground - test:ground	
train	test	mean accuracy	standard deviation	mean accuracy	standard deviation
272	279	0.981496	0.0519346	0.97619	0.0719677
180	371	0.965315	0.0810797	0.966667	0.11547
134	417	0.895425	0.249018	0.90873	0.286944
107	444	0.90078	0.203108	0.906629	0.285934
88	463	0.87253	0.253706	0.895028	0.232751
74	477	0.848693	0.247112	0.871555	0.237789
63	488	0.811254	0.321924	0.851051	0.285232
57	494	0.80694	0.333319	0.863545	0.276736
50	501	0.790199	0.312047	0.819091	0.271874

Table 4.6: Classification results using a linear Support Vector Machine for class labels predicted by SFA and the manually annotated ground truth labels for the extended tracks dataset. Even tough the SVM learned on a few wrong labels (see table 4.4) the classification accuracy is comparable to the ground truth accuracy.

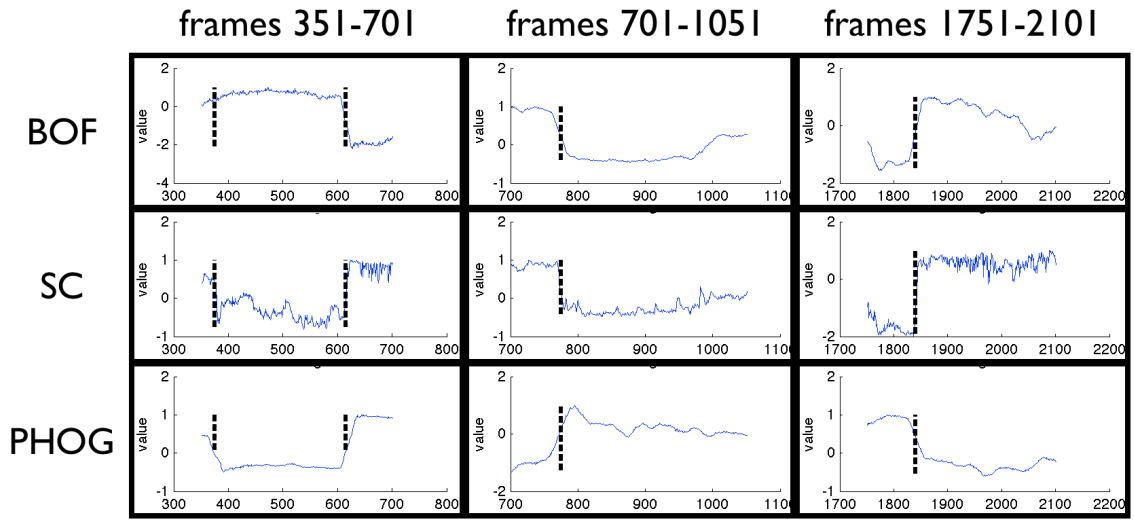

Figure 4.6: Slowly varying signals for different batches of frames and different feature descriptors (BOF= bag-of-features representations based on SIFT, SC= Sparse Coding, PHOG= Pyramid Histogram of Oriented Gradients). The ground truth class changes are marked with black dashed lines in the signals. All slow signals extracted from the different feature descriptors indicate some change, some more prominent than others.

On the other hand the extended tracks dataset contains video sequences from people crossing a road with clear cuts between different persons. However this is still a challenging dataset as it contains nearly the same background information for all classes and the SFA algorithm must be able to discriminate the classes without getting distracted by the constant background information.

For both datasets, the results for the supervised classification experiments show, that the classification accuracy drops when fewer training samples are used as it is expected for any classification algorithm. More interesting however is the fact that even for very few training samples the linear SVM can distinguish the individual classes based on the sparse feature descriptor very well. Even having only 16 individual samples (not 16 per class) to train the SVM , evaluating on the 4686 remaining samples gives an accuracy much better than random guessing. In both tables it is visible that the difference in classification capability if trained on the SFA labels or on the ground truth is under 1% for a reasonable amount of training samples.

Note that the amount of training samples is not a limiting factor for the proposed solution as labels for every frame are available and could be used in the training phase. It is therefore valid to state that the proposed way of performing supervised classification based on estimated class labels produces comparable results by gaining the additional benefit that it can also be performed on unlabeled video sequences.

Summarizing the whole chapter, including the outcome from various experiments, it contains a novel approach to automated labeling of video sequences based on slowness information extracted using slow feature analysis and Sparse Coding as robust mid-level feature representation. In contrast to other labeling methods based on tracking or self-learning, this method works without any initial setup, and is therefore not dependent on a correctly trained prior model. Using Sparse Coding as robust feature representation this approach can even handle huge changes in lighting, scale and rotation. The proposed method could successfully shrink the manual effort associated with generating labels for classifier training.

4.2 Unusual Event Detection

This section describes a problem setting where the sparse coding feature descriptor is used for the task of detecting unusual events in video sequences. The task of finding unusual events is shortly introduced in section 4.2.1 and section 4.2.2 reviews tree state-of-the art systems for detecting unusual events based on sparse coding, multiple local monitors and infinite Hidden Markov models.

Section 4.2.4 describes the implementation details and the performed experiments to evaluate the usability of Sparse Coding for such a task. Its results are discussed in section 4.2.5.

4.2.1 Problem Definition

The following section describes the concept of unusual events and the detection of such in different contexts and media, where first of all the ambiguity of the term “unusual event” itself will be treated [106] [10].

The first problem one faces when trying to generally define the concept of an unusual event is that it strongly depends on the context, but also on the definition of its counterpart: What is a usual event in the same context?

The application area of unusual event detection is widespread: it ranges from audio, to visual information like videos or images, but also to any other sensory data.

In most cases, unusual events are related to irregular and extraordinary, unlikely or unexpected, rare events in a sequence of normal ones. Usually they describe some temporal state, which is often more relevant than the more likely and expected regular events, like e.g. suspicious behavior in some video.

The reason why the detection of such irregular events gets important is related to the usual problem of having too much data, containing only few important informations - like in the field of surveillance. As already mentioned in the motivation, hours of video material is recorded every day, but the interesting and noteworthy moments of this data normally describe a

special or irregular event. Unusual event detection allows to detect such events, therefore acting as a filter to human control.

When applying unusual event detection, supervised methodologies often emerge to be not well suited: semi-supervised or even completely unsupervised techniques can iteratively learn or adapt to the changing "usual" situation, resulting in more reliable detections of unusual events.

For detecting such unusual events different approaches exist, some of them are briefly explained in the following section.

Figure 4.7: Some scenes from the UCSD Anomaly Detection Dataset: the first two are normal samples containing walking people, the abnormalities depicted are two biker, a skateboarder and a golf cart

4.2.2 Different Approaches to Detect Unusual Events

Unusual event detection using Sparse Coding

The system proposed by Zhao, Fei-Fei and Xing [108] detects unusual events in an online fashion using Sparse Coding. Its core idea is the same as for our system that is discussed in section 4.2.4.

Their system does not need any initialization phase where a model based on normal events is learned, therefore no initial training data is learned. Using Sparse Coding they compute a reconstruction cost which determines the absence or presence of an unusual event in the following way:

After having learned an initial dictionary needed for Sparse Coding using a sliding window approach each event is described using spatio-temporal interest points. For each such event the optimal reconstruction cost is calculated by essentially solving the Sparse Coding optimization problem with the current dictionary. If this cost lies above a certain threshold the current frame is signaled as unusual event. Before analyzing the next video frame the codebook is updated.

The idea behind this technique is that the reconstruction error of the unusual, irregular events is higher than the one of the majority of the video frames: the unusual events cannot easily be reconstructed, as the codebook will mostly contain elements to describe usual events.

In addition to the fact, that no initial knowledge about the content of the video is required, the completely unsupervised approach also enables continuous learning during the entire video, still not allowing for concept drifts.

Cong, Yuan and Liu describe a very similar approach in their work [22]. They introduce a sparse reconstruction cost, which corresponds to the reconstruction error used by Zhao et al, using this cost as a measure for the rarity (or normalness) of single events. The main difference lies in the choice of the used feature, where Cong et al rely on a Multi-scale Histogram of Optical Flow to describe the video scenes. Their initial system is defined for offline use, but in addition to that, the authors extend their model to also enable online detection of unusual events by continuous incremental updates of the dictionary.

Unusual event detection using multiple monitors

One approach of detecting unusual or rare events is to use multiple local monitors, as described in the paper of Adam et al [1].

They describe a large-scale surveillance system which is based on the idea of collecting local low-level statistics or measurements at multiple fixed spacial locations, called monitors. The idea is that if the current measurement of one of these monitors is unusual or abnormal, this single monitor gives an alert. Using some ambiguity threshold, the over-all system comes to a final decision whether there exists a local or global unusual event.

The technology described in this paper is applied to long-time real-life videos, as it is the case in large-scale surveillance according to the authors.

This method of using multiple monitors is quiet different, as its success or failure does not only depend on the implemented algorithm, but also on the setup of the system itself. Adam et al state for example, that in order to best possibly overcome the problem of occlusion, they use a density of the monitors such that each individual is usually “tracked” by five monitors.

An advantage of this technique is that it performs well and it is quiet robust also in crowded scenes because of the multiple spacial information that is collected contemporaneously, which appears to be a major problem of usual tracking-based algorithms, causing them to fail due to the crowdedness, rendering the tracking of single individuals nearly impossible.

According to the authors, the system is quiet effective after having collected sufficient low-level data, which is usually the case after a few minutes. In this context it has to be mentioned that most unusual event detectors are tested on relatively short video sequences, where the currently explained approach would fail, as this short time tests often do not even cover the time period required for the setup and the initial learning of this multiple monitor system.

On the other side, this approach seems to be well suited for many real-time applications: It is not computationally demanding and is tested on relatively long video sequences. Additionally, Adam et al state that the principle of having a simple and intuitive algorithm like in this case brings the advantage that the entire system gets predictable and thus its decisions can be better understood.

One drawback that the authors claim is that still it is not possible to detect unusual events if they are composed of a sequence of usual events, e.g. if someone enters a security door without previously having opened it by inserting an identification card.

Unusual event detection using infinite hidden Markov models

In contrast to other techniques, Zhang et al present an approach based on iteratively adapted hidden Markov models [77]. More precisely, they use ergodic K-class HMMs, where the number of classes is equal to one normal state plus the number of unusual events.

Their model is built up using one single state (or HMM class), which should represent the normal situation or event. This starting state is learned a priori, by using a set of usual event examples. The model, i.e. the parameters of this normal event, is learned using a maximum likelihood method on the usual events. The authors use a Gaussian mixture to model the probability density functions of all the HMMs. The corresponding parameters are estimated using expectation maximization.

After having learned this HMM representing the usual event, the test video is divided into overlapping sub-sequences, computing for each of the sub-sequences the likelihood, where low values indicate abnormal events.

As soon as such an unusual event is detected, this one is supposed to represent a new HMM, even though it is not possible to train and learn an entire model using only one single example (i.e. the currently detected unusual event). Therefore the authors propose to use some adaptation techniques, like maximum a posteriori, in order to adapt the existing model - representing the normal state - to the new unusual situation.

After knowing the models for the normal state (which has been well learned) and an irregular state (given by a currently detected unusual event), the HMM is modified, by adding an additional state to it. Viterbi decoding is applied to the updated HMM and the parameters, to detect the transition points and thus when the unusual event started (and ended).

Then the system proceeds, searching for the next lowest likelihood, and in this way detecting the next abnormal event in the sequence.

Zhang et al give results of testing their approach on audio (i.e. speech with applause, or laughter inside), on videos (a poker game, including cheating, like hiding cards), and on audio-visual data, i.e. a video about a presentation containing both kinds of information. They argue that their approach results in being able to successfully detect rare and unexpected events, after having been trained on an appropriate number of usual events, which in most practical cases is available.

4.2.3 Motivation

In the following I try to motivate why an unusual event detection system based on features developed using Sparse Coding might be an interesting area for experiments and what is expected from the results.

There exists plenty of literature [39, 10, 77, 108, 1, 106, 22] on abnormal event detection systems, each working in a slightly different environment or under different constraints. Most of them are evaluated on different datasets, using different methods to model usual or unusual events and even report different quality measures. Depending on the desired application scenario for some systems it is sufficient to reliably determine whether a specific time span contains usual or unusual events, while for others it might be necessary to also precisely locate the source of abnormality.

Many recent publications [107, 78, 19, 103, 104] in the area of computer vision show that sparse representation-based methods achieve good performance for various application domains. All applications have in common that they use a sparse mid-level feature representation of the task specific basis elements (e.g., motion primitives for activity recognition). Using an appropriate pooling function, Sparse Coding gives a compact and robust appearance description. Describing activities as sparse combinations of learned motion primitives has shown to result in state-of-the-art performance for activity recognition tasks [78, 104]. Unusual event detection can be seen as a simpler form of activity recognition, where, instead of recognizing a learned activity pattern, the task is simplified to detect whether an activity is close to the learned ones or not. In this case, the learned "activities" describe usual events, unusual events are then measured based on their relation to the learned ones. Since Sparse Coding has been shown to perform good for activity recognition, we expect that this sparse mid-level representation is also suited for the task of unusual event detection.

The first motivation to build this unusual event detection system based on Sparse Coding is therefore not only to evaluate Sparse Coding for this task, but also to provide a framework and platform to evaluate other feature descriptors, under the same conditions and on the same data. This results in a fair and comparable performance evaluation for different feature descriptors.

From this experiments we expect results indicating that Sparse Coding performs at least as good as other state-of-the-art appearance-based feature descriptors. This expectation is based on various recent publications where sparse feature descriptors could outperform other methods on object recognition tasks. Therefore we expect that some of the information modeled by the feature descriptor, useful for object recognition, might also prove useful for unusual event detection. Since abnormal event detection could be interpreted as a binary classification problem generating outputs "usual event" or "unusual event", this seams a reasonable assumption.

We also evaluate the performance of a motion-based feature descriptor (Histogram of oriented flow) and compare it to the performance of the other appearance-based descriptors. For this comparison we do not state any initial believes or thoughts about its outcome.

On one hand Sparse Coding describes each image based on its appearance, and we expect unusual events to show a different semblance than usual ones. On the other hand many unusual events might also expose different motion patterns than usual ones. Pedestrians and skateborders generate different motion while having a really similar appearance. Therefore this is an interesting comparison, allowing to express strengths or weaknesses of motion- vs. appearance-based feature descriptors for unusual event detection.

4.2.4 Implementation

The used system for detecting unusual events in video streams is comparable to the systems reviewed in the previous section 4.2.2. Generally the system proceeds in two stages: during learning stage usual events are analyzed to model usual events. During detection phase the modeled usual behavior is compared with the current video scene and its discrepancy is measured. Based on this distance measure the algorithm decides then whether to signal an unusual situation or not.

This basic architecture is now explained in more detail and then used later on to perform various experiments. An overview over the entire learning and detection process is additionally shown in figure 4.8.

As a first step when trying to learn usual events we need to represent the current video scene with some sort of feature descriptor, either raw pixel values or more advanced descriptors like SIFT [62] or HOG [23]. Since we aim to detect not only global unusual scenes but also the abnormality's location, we divide the input image into a fixed number of $4 \times 3 = 12$

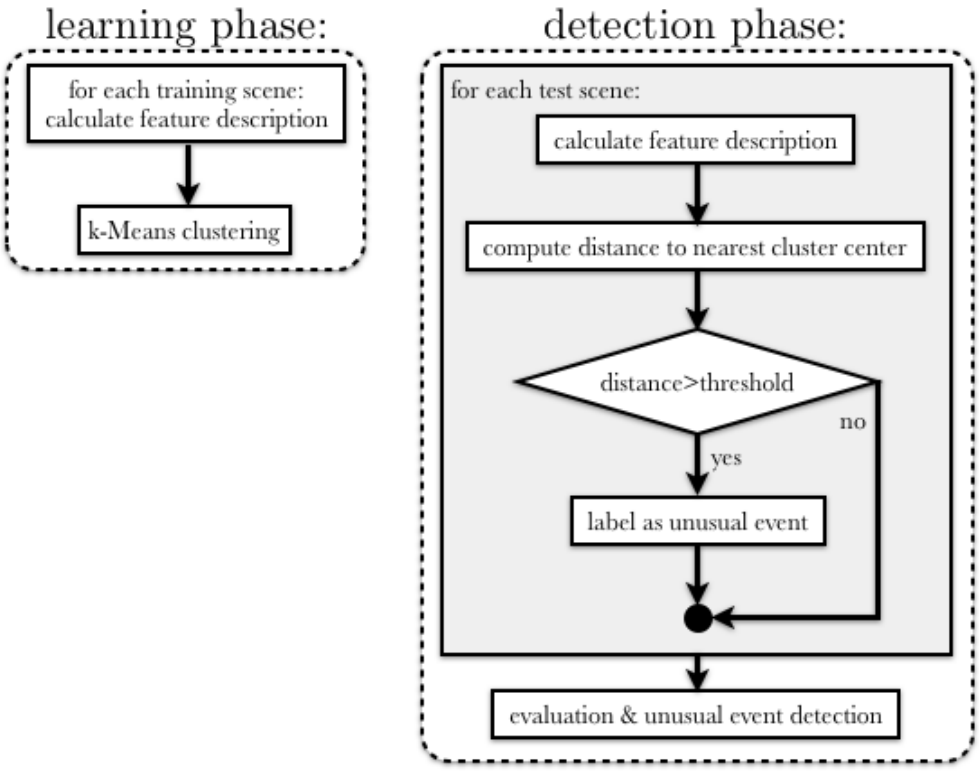

Figure 4.8: An overview over the process of detecting an unusual event: The left box shows the learning phase. On the right, the detection phase is sketched, where the single frames of the input video are checked for unusual events.

equaly-sized patches and compute the feature descriptors for each patch. This decomposition of the single input image into twelve image regions allows to model local normality and detect the unusual events within single patches whose position in the input image is known. An illustration of these patches can be seen in image 4.9, containing some sample results and two unusual events detected in different scenes.

One decision, sensitive to the final performance of the described system, is the choice of the feature descriptor to use to describe the video sequence. To evaluate and compare the performance of Sparse Coding several feature descriptors have been used which are now described shortly.

- **MV:** As a basic feature descriptor mean and variance of the pixel intensities within a single local patch are computed.
- **HOG:** The histogram of oriented gradients [23] which basically counts frequencies of gradient directions or edge orientations at fixed positions.
- **HOF:** The histogram of oriented flow [17] represents an image as optical flow information binned on its primary angle and weighted according to its magnitude.

- **SC:** The Sparse Coding feature descriptor has been presented in chapter 3.
- **BOW-SIFT:** Describes a bag-of-words model computed on SIFT features.

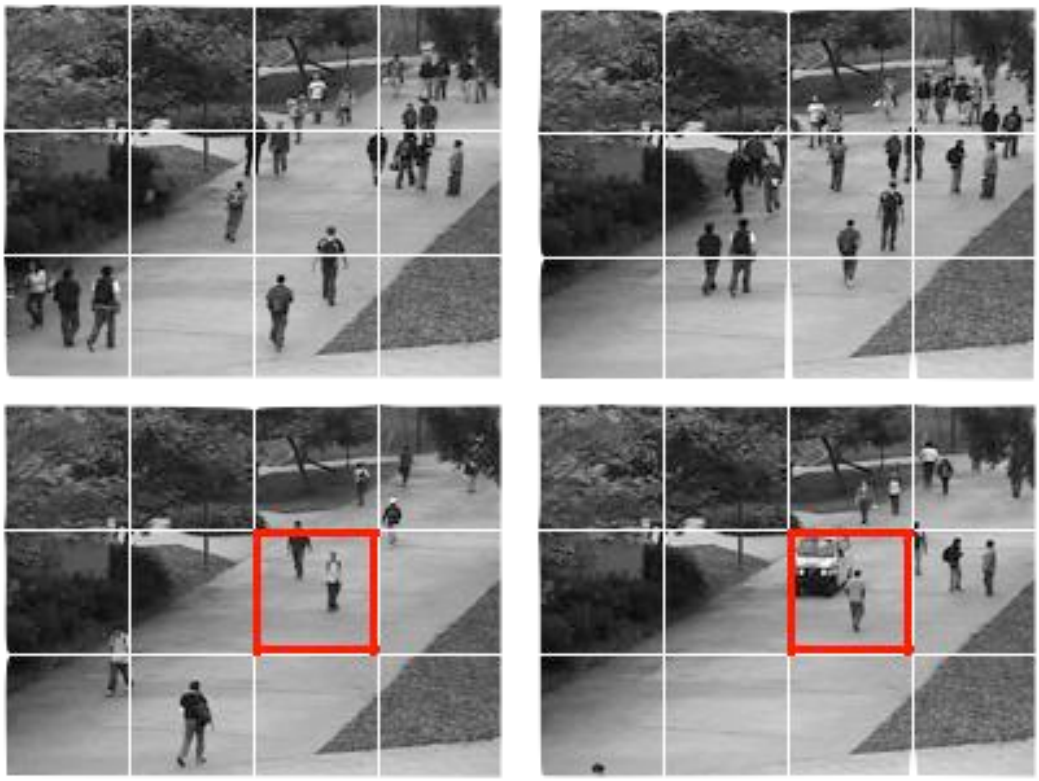

Figure 4.9: Four images illustrating some results for the unusual event detection on the UCSD Anomaly Detection dataset: the top images contain no unusual events where all people are walking. The images at the bottom contain unusual events: one contains a skateboarder, the other one a golf car.

Summarizing these features it is to note that they express quite different characteristics. The mean-variance feature describes each local patch as mean and variance of the greyscale pixel values. The intuition to use and evaluate such a simple feature descriptor is that even the pixel intensities should change when representing unusual events and therefore its variance should be high compared to the learned usual pixel intensities. In contrast HOG is a more advanced feature descriptor which is robust to most photometric and geometric transformations. With HOF a scene (containing either usual or unusual events) is represented as optical flow or motion of objects surfaces and edges within the scene. This feature descriptor is used to analyze if usual and unusual scenes differ significantly in their optical flow and can therefore be detected and separated. For the Sparse Coding feature descriptor we assume that usual and unusual events are visually different and therefore different codebook entries are used

when computing the sparse feature descriptors. Therefore the feature descriptors describing unusual events are expected to be different enough to be separated from usual events.

For each of the already mentioned local image patches the feature descriptors are calculated and stored to be used in the next step. In the case of sparse coding the exact pipeline as described in 3.3 has been followed. For the final implementation, a 1024-dimensional codebook pretrained on the Caltech101 dataset has been used although different codebooks and sizes have been evaluated. Experiments with a codebook created on the video sequence used for the unusual event detection task did not give any markable performance change as also supported by the experiments in 3.6.3 which show that a generic codebook can give comparable performance. The actual training stage is a computation of clusters based on all training sequences containing usual events. These clusters are calculated for each of the 12 image regions separately, therefore to each local cell up to 500 clusters are assigned. These clusters represent the most dominant features used to describe usual events within a specific location. For the actual clustering the fast implementation of the k-means algorithm from Piotr Dollár's Matlab toolbox[2] has been used.

The last step missing is the detection of the actual unusual events. For each video frame from all video sequences within the test set, the distance of its associated features with the previously learned centroids is calculated and the minimal distance from a feature to a centroid is used as measure of compliance with the previously learned model. The rational behind this is that features close to the one seen in the learning phase (or its class centers respectively) are more likely to not represent unusual events. After temporal smoothing over two consecutive image frames to prevent unwanted spikes in the distance for each of the 12 image locations, the presence or absence of an unusual event is decided by a simple thresholding criterion.

4.2.5 Results and Discussion

To evaluate the performance of a sparse feature descriptor, experiments have been conducted to compare it against other feature descriptors using the same detection framework as described in section 4.2.4.

As dataset, the UCSD Anomaly Detection Dataset, created for the publication [63] and containing crowded public walkways has been used. It has been generated using stationary cameras observing public walkways, containing various crowdedness levels. As normal events, this dataset contains people walking on the walkways. Abnormal events are characterized by people using other means of transportation (bikes, wheelchairs, skateboards, golf cars) or people walking on the green spaces instead of the walkway. The recorded video material has been splitter into various short scenes, each containing approximately 200 frames. Included in the dataset is also a manually generated ground truth dataset. For each frame the ground truth data defines if a unusual event is present or not. In addition for 10 sequences the exact location of the abnormality is represented as binary pixel mask.

[2] `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`

This dataset has been chosen for this experiments because it contains a wide variety of unusual events. Some of them my be easily detected by its appearance (golf car) while others are very similar to normal walking people, but expose a different motion pattern. This variety allows us to reason which unusual events are easier or harder to detect and why Sparse Coding might serve as a good representation to encode them.

As basic feature descriptors the following well-known and already introduced feature descriptors have been used: Sparse Coding, HOF, HOG, MV and BOW-SIFT. For each feature descriptor various detection thresholds were tested and, based on the available ground truth from the UCSD dataset, the number of detected true positive, true negative, false positive and false negative unusual events were recorded on a per-frame basis.

Based on this data, the performance is calculated as precision, recall and F1 score. Precision is defined as

$$precision = \frac{truepositive}{truepositive + falsepositive}$$

and gives a ratio of how many of the detected unusual events are really unusual events. Recall on the other hand measures the ratio of true unusual events within the detected ones and is defined as

$$recall = \frac{truepositive}{truepositive + falsenegative}$$

. It is easy to show that both of these measures influence each other: suppose our unusual event detector just detects one single frame as unusual event, all other events are recognized as usual. This gives the maximal precision of 1 but would lead to a lower recall (assuming that there is more than one unusual frame within the observed sequence). On the other hand one could achieve high recall by detecting all frames as unusual events, leading to 0 false negative samples and therefore a recall of 1, but also to a low precision due to the high number of false positives.

Therefore for most problems it is needed to balance those two measures, where for some problems it might be favorable to trade recall for additional precision, while for others it is not. One measure that balances both precision and recall is the so-called F-score which is used as final performance indicator for this experiments. It is defined as

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

and for our case using $\beta = 1$ (often also called F1-score) gives

$$F_\beta = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

. It defines a weighted average between recall and precision and ranges from 0 to 1.

The best performance expressed as F1 score for the different feature descriptors on 9 different video sequences taken from the UCSD dataset is reported in table 4.7.

	19	20	24	25	26	27	31	33	36
SC	**0,666**	**0,812**	**0,830**	**0,725**	0,511	**0,821**	**0,946**	**0,897**	**0,664**
HOF	0,608	0,793	0,786	0,650	0,509	0,724	0,944	0,894	0,641
HOG	0,337	0,451	0,030	0,518	0,083	0,511	0	0,207	0
MV	0,410	0,432	0,703	0	0	0,325	0,125	0	0,174
BOW	0,509	0,481	0,809	0,623	**0,602**	0,697	0,502	0,788	0,646s

Table 4.7: F1 measure for detecting unusual events on different UCSD video sequences for different feature descriptors.

The same data is also depicted in plot 4.10 and allows a few interpretations. First of all the data shows that the Histogramm of Oriented Flow, the Bag-of-Words model of SIFT features and the Sparse Coding feature descriptor are the ones performing best on the tested sequences. Even though the Mean-Variance descriptor performed surprisingly well on sequence 24 HOG and MV feature descriptors do not allow a reliable detection of unusual events for all tested sequences.

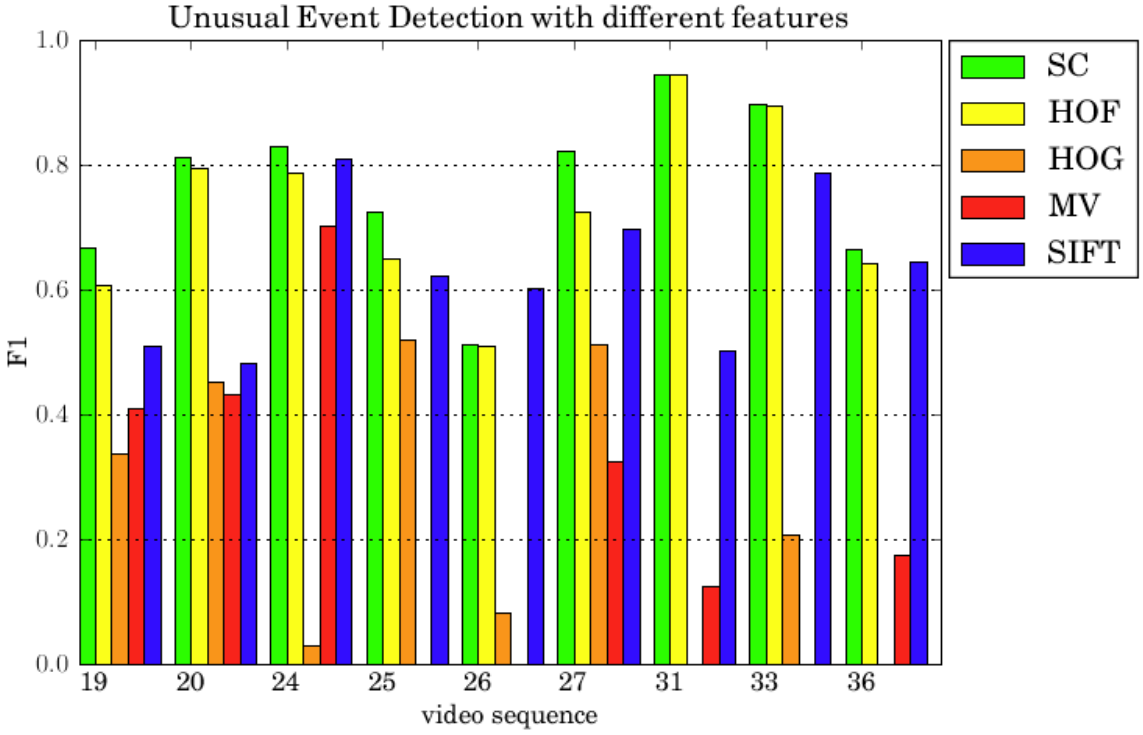

Figure 4.10: F1 score for nine different scenes taken from the UCSD Anomaly Detection dataset for different feature descriptors.

Except for video sequence 26 the optimal threshold for Sparse Coding always resulted in a higher F1 score than the other optimal configurations. For video 26 the bag-of-words model performed slightly better than the Sparse Coding descriptor.

Another interesting finding is that the performance of the Histogram of Oriented Flow is always very close to the one of Sparse Coding. If the one drops, the other one does, to. This is surprising because they describe the input using a totally different paradigm: HOF describes each image as a histogram of oriented optical movement, whereas sparse coding describes the scene as linear combination of a few heavy active base elements. This leads to the conclusion that for our image sequences a unusual event characterizes by an appearance change (encoded using sparse coding) and a movement change (encoded using optical flow). This seems intuitive, as most unusual events (golf car, wheelchair, bike) in our videos have different shapes with respect to walking people and also expose different motion behavior.

In the following a few scenes from different test videos completed with the detected unusual events for different feature descriptors are presented and discussed. This should give an intuition and better understanding of the meaning of the various performance indicators in our use case. It also should highlight individual strengths and weaknesses in an understandable format. Each image contains the same scene for the five feature descriptors, its 12 local blocks and, if detected, the red bordered cell containing an unusual event.

Figure 4.11 contains the results for detecting the golf car from UCSD scene 19 where the systems based on SC, HOG and HOF features did correctly identify the location of the unusual event. The systems using the bag-of-words feature and the mean-variance descriptor did not report any unusual behavior in this specific moment. It is also to note that the flow-based system contains two false-positive local alerts. This could be explained by the fact that for most training scenes both locations contained people and therefore movement while for the depicted frame there seems to be almost no movement in the foreground. This could therefore result in a high distance to learned movement patterns and therefore to a unusual event detection.

Figure 4.11: Detected unusual events for different descriptors based on UCSD scene 19

Images 4.12 and 4.13 both show again the detection results for unusual events contained in scenes 20 and 24 respectively. For those two frames the SIFT- and MV-based systems could only detect one of the two unusual events, while all other systems could correctly identify both driving cars as unusual with respect to the rest of the scenes. Again it is to note that some false positives where created, again most the motion feature.

Figure 4.12: Detected unusual events for different descriptors based on UCSD scene 20

Figure 4.13: Detected unusual events for different descriptors based on UCSD scene 24

The next two images contain results for more advanced, harder to detect unusual events. Figure 4.14 contains a bicyclist moving beside the pedestrian, both on approximately the same speed. In figure 4.15 there are two unusual events at the same time: a car driving down the walkway and a cyclist driving in the opposite direction.

For both scenes HOG and MV did not report any unusual events. Compared to the already described figures, in this two the histogram of oriented flow based system was able to detect all unusual events without any false positives being reported. The sparse coding based system could identify two of the three unusual events, but had problems on the cyclist which looks nearly identical to a usual pedestrian. This behavior could also be observed in other scenes and is based on the fact that both, a walking person and a cyclist from behind look similar and could be represented by the same basis elements from the dictionary, thus leading to similar feature representations. The bag-of-words based system performed well on the second of the last depicted scene, but reported only false positives on the previous one. For the Sparse Coding feature descriptors different combinations of extraction parameters (dense SIFT locations) and codebooks have been tested and confirm the results presented in section 3.6. We could not find any significant performance boost when exchanging the pre-trained

codebook based on the Caltech-101 dataset by one learned from the UCSD training dataset. For the perfect sampling size the intuitive results indicate that the performance gets better for finer sampling and significantly drops above a sample-distance of around 8 pixels. This can be well-explained because of the low-resolution images presented in the dataset: computing the SIFT features only every 8 or more cells results in only very few features per local patch, loosing most of the needed information.

Figure 4.14: Detected unusual events for different descriptors based on UCSD scene 36

Figure 4.15: Detected unusual events for different descriptors based on UCSD scene 36

Summarizing the findings we can argue that Sparse Coding achieves the best performance among all tested feature descriptors in the described setting. It could detect most of the unusual events without generating a huge amount of false positives. The Histogram of oriented flow and the bag-of-words SIFT descriptor performed also well but the motion feature tends to generate a lot of false positives in other cells. Depending whether a global or local unusual event detection system should be realized this has a lower or higher impact on the final result.

The results clearly indicate that a sparse mid-level representation generated via Sparse Coding allows better discrimination of usual from unusual events than low-level SIFT or HOG descriptors. The resulting video sequences show that the sparse representation is more robust to short occlusions (e.g., other pedestrians hiding the source to the unusual event) and can even detect unusual events when the motion-based feature is not able to express any abnormality.

5 Conclusion

Recently it has been shown that sparse, mid-level feature representations achieve state-of-the-art performance on many computer vision tasks. Although they are computationally more demanding than low-level feature descriptors, their robustness to great appearance changes justifies their usage. In addition, their sparse and compact representation achieves good results for many classification tasks. However, most of those publications where restricted to image-based methods. Therefore the goal of this master's thesis was to assess the applicability of Sparse Coding for video-based applications.

The information presented and elaborated in this master's thesis focused on Sparse Coding and its applicability and performance for analyzing image sequences. After a short introduction to the notion of feature descriptors in computer vision the main aspects of Sparse Coding have been presented. Sparse coding has been derived as a convex optimization problem starting from a general matrix factorization problem, including efficient algorithms to solve it. The whole coding pipeline starting from a raw input image until the computation of the final global sparse feature descriptor has been presented and analyzed.

Studies on the impact of codebook size and quality have been performed and indicate that even codebooks trained on a completely different, task-unrelated dataset can provide the same performance as using the task-specific dataset. This knowledge about the needed codebook quality a high range of flexibility when performing experiments with Sparse Coding, as it is not necessary to train a codebook on the given dataset to guarantee good results. Codebooks can therefore be shared and reused in other experiments.

The data also shows that codebook size is not a critical parameter, as Sparse Coding even performs good with very small codebooks. However performance starts to drop when the codebook grows too large, due to the noise modeled into the codebook. This implies that when performing Sparse Coding, the codebooks size should be kept within limits to not create noisy descriptions.

The next experiments confirmed that the biologically plausible maximum-pooling operation results in the best-performing sparse codes compared to other pooling functions and should thus be considered for various tasks dealing with Sparse Coding.

After this analysis of sparse coding two different video-based applications have been elaborated. The work published in [86] is described in much detail, and in this way a novel approach for automated labeling of video sequences based on a Slow Feature Analysis performed on the Sparse Coding feature descriptor is presented. It allows to automatically generate a

large amount of labels, and experimental results show that the labeling process creates good results. It has also been shown that even though feature descriptors can change heavily from one image frame to the next, slow feature analysis provides a powerful framework to extract and analyze the underlying slow driving force. The outcome encourage to use the proposed system in real-world applications, like automatically segmenting and labeling huge media catalogs from video-on-demand services.

The second developed application tackled the task of unusual event detection within video sequences using Sparse Coding as efficient and compact feature representation. Its performance has been evaluated and compared to different structure and motion based feature descriptors. Experimental results show that it generates better performance than other state-of-the-art appearance-based descriptors and also creates less false alarms than the motion-based Histogram of Oriented Motion feature. It would thus be a readonable decision to use Sparse Coding for unusual event detection in various fields of applications, like video surveillance.

Intuitively this master's thesis is not able to answer all possible research questions in the presented field. But the conducted experiments and results opened possibilities for future work and new research questions.

For Sparse Coding in general an interesting opportunity for future research could try to come up with a real online variant of the whole coding pipeline. Although algorithms for online codebook learning exist, the whole feature computation is still not possible in an online fashion. Since for every new input image the codebook is updated in an online fashion, all preceding feature descriptors are based on outdated information and would need a projection onto the new codebook. With the availability of such an online method, the codebook refinement and feature computation could be performed simultaneously without the need of recomputing all previous features.

The system from [86] contains an exemplary application and demonstration of slowness as a feature descriptor and stimulates further work in this direction. It would be preferable not to use the slow signal directly for image labeling, but to use it as an unsupervised class prior for various detection or tracking tasks. Modeling this slowness into a semi-supervised setting could allow to detect similar objects based on the slowness characteristics, despite the fact that their feature descriptors are totally different.

The described unusual event detection system illustrates strengths and weaknesses of the feature descriptors' expressiveness which could be verified on related tasks. An experiment worth to perform would be the combination of appearance and motion to detect unusual events, as some scenes have shown to contain events hard to detect for motion-based systems but easy for the ones operating on appearance information and vice-versa. This could also allow to actually learn the coherence of motion and appearance in such systems.

The main contributions in this work are the following:

- An indeed analysis of Sparse Coding, its theoretical foundations and algorithmic challenges, and the main parameters has been performed. This work stresses the importance of spatial pooling to create a robust representation, and highlights the importance of the codebook. The biologically plausible maximum and average spatial pooling functions have shown to provide the best results. Additionally, this work suggests that codebooks containing between 256 and 2048 atoms should be used for best performance.
- This work presented a novel approach to unsupervised image sequence labeling by analyzing the mid-level feature representation over time [86]. This analysis has been computed using Slow Feature Analysis, showing that a slow driving force can be used to efficiently detect visual class changes. Using Sparse Coding as image representation, larger appearance changes than with traditional methods can be handled. In addition it stimulates the usage of Slow Feature Analysis for other image-sequence based applications that aim at analyzing descriptors over time.
- As part of this master's thesis it has also been shown, that Sparse Coding is a powerful feature representation for the task of unusual event detection. It could outperform all other tested appearance-based descriptors and generated less false positives than the motion based feature.

Bibliography

[1] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz. Robust real-time unusual event detection using multiple fixed-location monitors. *PAMI*, 2008.

[2] J. K. Aggarwal and Q. Cai. Human motion analysis: A review. *Computer Vision and Image Understanding*, 1999.

[3] M. Andriluka, S. Roth, and B. Schiele. People tracking-by-detection and people-detection-by-tracking. *CVPR*, 2008.

[4] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. *Computer Vision and Image Understanding*, 2008.

[5] M. Bennamoun and G. J. Mamic. *Object recognition: fundamentals and case studies*. Springer, 2002.

[6] P. Berkes. sfa-tk: Slow feature analysis toolkit for matlab, 2003.

[7] P. Berkes. Pattern recognition with slow feature analysis, 2005.

[8] M. Berry, M. Browne, A. Langville, V. Pauca, and R. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics And Data Analysis*, 2007.

[9] T. Blaschke, P. Berkes, and L. Wiskott. What is the relation between slow feature analysis and independent component analysis? *Neural Computation*, 2006.

[10] O. Boiman and M. Irani. Detecting irregularities in images and in video. *International Journal of Computer Vision*, 2007.

[11] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *CVPR*, 2008.

[12] A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. In *ICCV*, 2007.

[13] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. *CVPR*, 2007.

[14] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*, 2010.

[15] Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *ICML*, 2010.

[16] S. Boyd and L. Vandenberghe. *Convex Optimization*, volume 7. Cambridge University Press, 2009.

[17] R. Chaudhry, A. Ravichandran, G. Hager, and R. Vidal. Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. *CVPR*, 2009.

[18] C.-C. Chen and J. K. Aggarwal. Recognizing human action from a far field of view. *IEEE Workshop on Motion and Video Computing*, 2009.

[19] C.-k. Chiang, C.-h. Duan, and S.-h. Lai. Learning component-level sparse representation using histogram information for image classification. *CVPR*, 2011.

[20] A. Coates and A. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, 2011.

[21] P. Comon and C. Jutten. *Handbook of Blind Source Separation: Independent Component Analysis and Applications*. Academic Press, 1st edition, 2010.

[22] Y. Cong, J. Yuan, and J. Liu. Sparse reconstruction cost for abnormal event detection. *CVPR*, 2011.

[23] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *CVPR*, 2005.

[24] R. Donner, E. Birngruber, H. Steiner, H. Bischof, and G. Langs. Localization of 3d anatomical structures using random forests and discrete optimization. In *MICCAI*, 2011.

[25] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley and Sons, 2001.

[26] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 2007.

[27] P. Földiák and M. P. Young. Sparse coding in the primate cortex. In *The Handbook of Brain Theory and Neural Networks*, 1995.

[28] M. Franzius, N. Wilbert, and L. Wiskott. Invariant object recognition with slow feature analysis. *ICANN*, 2008.

[29] M. J. Gangeh, L. Sørensen, S. B. Shaker, M. S. Kamel, M. De Bruijne, and M. Loog. A texton-based approach for the classification of lung parenchyma in ct images. *Medical Image Computing and Computer-Assisted Intervention*, 2010.

[30] T. J. Gawne and J. M. Martin. Responses of primate visual cortical V4 neurons to simultaneously presented stimuli. *Journal of Neurophysiology*, 2002.

[31] J. E. Gentle. *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer, 2010.

[32] G. H. Golub and C. F. Van Loan. *Matrix Computation*. The Johns Hopkins University Press, 3rd edition, Oct. 1996.

[33] E. F. Gonzalez and Y. Zhang. Accelerating the lee-seung algorithm for nonnegative matrix factorization. Technical Report TR05-02, 2005.

[34] M. Grabner, H. Grabner, and H. Bischof. Fast approximated sift. *Asian Conference on Computer Vision*, 2006.

[35] K. Gregor and Y. Lecun. Learning fast approximations of sparse coding. *ICML*, 2010.

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

[37] R. Grosse, R. Raina, H. Kwong, and A. Y. Ng. Shift-invariant sparse coding for audio classification. In *Uncertainty in Artificial Intelligence*. Citeseer, 2007.

[38] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Westview Press, 1991.

[39] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 2004.

[40] P. O. Hoyer. Non-negative sparse coding. In *IEEE Workshop on Neural Networks for Signal Processing*, 2002.

[41] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009.

[42] R. L. Kashyap. Video scene change detection method using unsupervised segmentation and object tracking. *ICME*, 2001.

[43] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *CVPR*, 2004.

[44] S. Klampfl and W. Maass. A theoretical basis for emergent pattern discrimination in neural systems through slow feature extraction. *Neural Computation*, 2009.

[45] V. R. Kompella and M. Luciw. Incremental slow feature analysis. *IJCAI*, 2011.

[46] W. Konen. On the numeric stability of the SFA implementation sfa-tk. Technical report, Cologne University of Applied Sciences, 2009.

[47] W. Konen and P. Koch. How slow is slow? sfa detects signals that are slower than the driving force. Technical report, Cologne University of Applied Sciences, 2009.

[48] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[49] T. Kühnl, F. Kummert, and J. Fritsch. Monocular road segmentation using slow feature analysis. *Intelligent Vehicles Symposium*, 2011.

[50] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *CVPR*, 2006.

[51] Y. Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1990.

[52] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 1999.

[53] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2000.

[54] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances NIPS*, 2006.

[55] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, 2009.

[56] D. J. Leggett. Numerical analysis of multicomponent spectra. *Analytical Chemistry*, 1977.

[57] D. Lelescu and D. Schonfeld. Statistical sequential analysis for real-time video scene change detection on compressed multimedia bitstream. *IEEE Transactions on Multimedia*, 2003.

[58] F. Li, J. Carreira, and C. Sminchisescu. Object recognition as ranking holistic figure-ground hypotheses. In *CVPR*, 2010.

[59] X. Li. Hmm based action recognition using oriented histograms of optical flow field. *Electronics Letters*, 2007.

[60] P. Lipson, A. Yuille, D. O'Keeffe, J. Cavanaugh, J. Taaffe, and D. Rosenthal. Deformable templates for feature extraction from medical images. *ECCV*, 1990.

[61] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[62] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[63] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos. Anomaly detection in crowded scenes. In *CVPR*, 2010.

[64] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *ICML*, 2009.

[65] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 2010.

[66] J. Mairal, M. Elad, and G. Sapiro. Sparse representation for color image restoration. *IEEE Transactions on Image Processing*, 2008.

[67] L. Matthies, M. Maimone, A. Johnson, Y. Cheng, R. Willson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein, and A. Angelova. Computer vision on mars. *IJCV*, 2007.

[68] J. Mutch and D. Lowe. Multiclass object recognition with sparse, localized features. *CVPR*, 2006.

[69] J. Mutch and D. G. Lowe. Object class recognition and localization using sparse features with limited receptive fields. *IJCV*, 2008.

[70] Nayar and H. Murase. Columbia object image library: Coil-100. Technical Report CUCS-006-96, Department of Computer Science, Columbia University, February 1996.

[71] C.-w. Ngo, T. Hong, and C. W. Bay. Motion-based video representation for scene change detection. *ICPR*, 2002.

[72] H. Nickisch. Extraction of visual features from natural video data using slow feature analysis. Master's thesis, Technische Universität Berlin, 2006.

[73] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *ECCV*, 2006.

[74] B. Olshausen and D. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 1996.

[75] B. Olshausen and D. Field. Natural image statistics and efficient coding. In *Computation in Neural Systems*, 1996.

[76] P. Ott. Incremental matrix factorization for collaborative filtering. *Contributions to Science, Technology and Design*, 2008.

[77] I. Pruteanu-Malinici and L. Carin. Infinite hidden markov models for unusual-event detection in video. *IEEE Transactions on Image Processing*, 2008.

[78] Q. Qiu, Z. Jiang, and R. Chellappa. Sparse dictionary-based representation and recognition of action attributes. *CVPR*, 2011.

[79] M. Ranzato, Y.-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *NIPS*, 2007.

[80] M. A. Ranzato, F. J. Huang, Y. L. Boureau, and Y. Lecun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *CVPR*, 2007.

[81] P. M. Roth, T. Mauthner, I. Khan, and H. Bischof. Efficient human action recognition by cascaded linear classifcation. *ICCV*, 2009.

[82] S. Savarese, J. Winn, and A. Criminisi. Discriminative object class models of appearance and shape by correlatons. *CVPR*, 2006.

[83] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *CVPR*, 2005.

[84] B. Shen and C. H. Chen. Bhattacharyya distance based video scene change detection. *Pattern Recognition*, 2009.

[85] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.

[86] M. Stricker, P. M. Roth, and H. Bischof. Slow feature analysis for autonomous learning object categories from videos. In *Computer Vision Winter Workshop*, 2012.

[87] E. B. Sudderth, A. Torralba, W. T. Freeman, and A. S. Willsky. Learning hierarchical models of scenes, objects, and parts. *ICCV*, 2005.

[88] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[89] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 2009.

[90] L. H. Y. Tang, R. Hanka, and H. H. S. Ip. A review of intelligent content-based indexing and browsing of medical images. *Health Informatics Journal*, 1999.

[91] S. Todorovic and N. Ahuja. Learning subcategory relevances for category recognition. In *CVPR*, 2008.

[92] M. Treiber. *An Introduction to Object Recognition*. Springer, 2010.

[93] R. Turner. A maximum-likelihood interpretation for slow feature analysis. *Neural Computation*, 2005.

[94] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 2007.

[95] T. Virtanen. Sound source separation using sparse coding with temporal continuity objective. *Proceedings of the International Computer Music Conference*, 2003.

[96] J. Wang, J. Yang, K. Yu, F. Lv, T. S. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *CVPR*, 2010.

[97] C. R. Weisbin, J. Blitch, D. Lavery, E. Krotkov, C. Shoemaker, L. Matthies, and G. Rodriguez. Miniature robots for space and military missions. *IEEE Robotics and Automation Magazine*, 1999.

[98] L. Wiskott and T. J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 2002.

[99] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *PAMI*, 2009.

[100] B. Xie, M. Song, and D. Tao. Large-scale dictionary learning for local coordinate coding. In *BMVC*, 2010.

[101] J. Yang, Y. Li, Y. Tian, L. Duan, and W. Gao. Group-sensitive multiple kernel learning for object categorization. In *ICCV*, 2009.

[102] J. Yang, K. Yu, Y. Gong, and T. S. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009.

[103] M. Yang and D. Zhang. Robust sparse coding for face recognition. *CVPR*, 2011.

[104] B. Yao, X. Jiang, A. Khosla, A. L. Lin, L. Guibas, and L. Fei-Fei. Human action recognition by learning bases of action attributes and parts. *ICCV*, 2011.

[105] X. Yi and N. Ling. Fast pixel-based video scene change detection. *Circuits ans Systems*, 2005.

[106] D. Zhang, D. Gatica-Perez, S. Bengio, and I. McCowan. Semi-supervised adapted hmms for unusual event detection. *CVPR*, 2005.

[107] L. Zhang, M. Yang, X. Feng, and H. Kong. Sparse representation or collaborative representation: Which helps face recognition? *CVPR*, 2011.

[108] B. Zhao and E. P. Xing. Online detection of unusual events in videos via dynamic sparse coding. *CVPR*, 2011.

[109] X. Zhou, K. Yu, T. Zhang, and T. S. Huang. Image classification using super-vector coding of local image descriptors. *ECCV*, 2010.

List of Figures

List of Tables

List of Algorithms