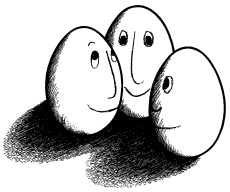
Diplomarbeit

Real-time feature extraction from video stream data for stream segmentation and tagging

Matthias Schulte



Diplomarbeit
an der Technischen Universität Dortmund
Fakultät für Informatik
Lehrstuhl für künstliche Intelligenz (LS8)
http://www-ai.cs.tu-dortmund.de

Dortmund, January 22, 2013

Betreuer:

Prof. Dr. Katharina Morik
Dipl.-Inform. Christian Bockermann

Acknowledgment

I wish to express my sincere gratitude to my supervising tutors Prof. Dr. Katharina Morik and Dipl.-Inform. Christian Bockermann for their qualified support. During the last months they have been available almost twenty-four-seven and helped me a lot by providing me with all the advice and practical support I required.

In addition I own thanks to everybody else working at the chair for artificial intelligence at TU Dortmund University. Beside supporting me with their scientific expertise, they always had a sympathetic ear and made me feel motivated and encouraged every time I was in the office.

Furthermore I would like to thank Radim Burget, author of the RapidMiner Image Extension, who helped me by sharing his code and supporting me whenever I had questions regarding the extension. It was a pleasure to work with you.

Last but not least I especially owe thanks to my beloved family and friends, who supported me in writing this thesis.

Contents

List of Figures

List of Tables

1

Introduction

Beside traditional Internet applications like world wide web and mail, the Internet becomes increasingly more interesting for broadband applications like video and television streaming. The *Cisco* Visual Networking Index Forecast (VNI), an ongoing initiative to track and forecast the impact of visual networking applications, estimates that in 2016 Internet video traffic will be 55 percent of all consumer Internet traffic [Cisco, 2012]. The video platform *YouTube*[1], founded in early 2005, solely accounted for 10 percent of all traffic on the Internet in 2007 and is continuously growing. Their website is visited by 800 million unique users per month, watching over four billion videos daily [YouTube, 2012]. In addition to video-on-demand services like *YouTube*, IP-TV (Internet Protocol Television) is rapidly growing as well. While only big events like sport events were broadcasted via IP-TV in the past, nowadays companies like *Zattoo*[2] offer the opportunity to watch a wide range of TV channels anywhere and anytime live over the Internet. The number of people using this service is continuously increasing: By June 2011 more than eight million users were registered on the *Zattoo*-Website [onlinekosten.de GmbH, 2012]. Within just a year this number increased to over ten million users by July 2012 [Zattoo, 2012]. Although not all users are active users, more than one million IP-TV streams per month are watched over the *Zattoo* platform.

Main reasons for watching IP-TV, either by using streaming services on the computer or by using IP-TV set-top-boxes, are probably:

- Given that an Internet connection is available, additional expenses for a cable connection or a satellite dish are omitted.

- Television becomes available on more devices such as computers, smart phones and tablets. Hence existing devises can be used for television reception and expenses for television sets are omitted as well.

[1] http://www.youtube.com
[2] http://zattoo.com

- Digital transmission allows better video and sound quality (HDTV).
- The number of potential channels is larger as no additional efforts have to be undertaken to receive the television programs of foreign stations.
- In many cases recording and video-on-demand functionality is already integrated and therefore free as well.

But IP-TV has even more advantages beside these obvious ones. The upcoming IP-TV "has begun to bring consumers (a) new TV experience which goes beyond any traditional passive TV" [Zeadally et al., 2011]. Reason for this is that IP-TV, contrary to the traditional TV, provides a feedback channel which allows interactivity with and personalization of the TV program. By tracking the switching behavior of users over the feedback channel, IP-TV providers can discover, what ever single customer is watching and fit the program to the customers' needs. This includes for instance a personalized recommendation of shows or an adaption of advertisement to the customers interest. However not only the customer but also the content providers profit. By fitting advertisement to the users needs, users are more likely to really buy the advertised products. Thus the value of the advertisement increases and hence content providers can earn more money without increasing the amount of advertisement. Furthermore the expensive and imprecise evaluation of user behavior by questionnaires or tracking the switching behavior of some random and representative households by using set-top box based data becomes redundant.

The above mentioned advantages require an extensive analysis of the gained user and video data. This includes on the one hand the enrichment of the data by all information available and on the other hand the analysis of this enriched data with machine learning approaches. As this analysis is as well of high scientific as of economic interest, big efforts have been made in the past twenty years to cope with the problem. These efforts include the extraction of features from audio and video data, the automatic parsing, segmenting, indexing and tagging of video content and the development of recommendation approaches suitable for video data based on user behavior. As tasks are various, scientists from many fields of research (computer scientists specialized on vision, machine learning or recommendation systems, mathematics and psychologists) have contributed and results are manifold. Nevertheless there is no existing overall system for real-time program recommendations based on user- and video-data. The new European Union-funded collaborative research project ViSTA-TV is addressing this problem.

The results of this thesis are supposed to be used within the ViSTA-TV project. By extracting real-time features from stream video data, we gain further knowledge about the video stream itself and thereby improve the quality of the data. The idea is, that the enriched data helps to built a recommendation engine. The next section is going to provide the reader with some basic information about the project.

1.1. The ViSTA-TV project

The project "Video Stream Analytics for Viewers in the TV Industry" (short: ViSTA-TV)[3] is a European Union-funded collaborative research project, beginning on June 1st, 2012, and lasting for two years. Its main objective is to bootstrap the IP-TV economy by enriching video content with background information, that allows to study user behavior and determine recommendations for users by predicting, which currently broadcasted show might be interesting for them.

Figure 1.1.: Logo of the ViSTA-TV project.

To achieve this objective, the main tasks of the ViSTA-TV project are

- to process real-time TV data (video-, user behavior- and TV meta-data) and extract useful features from this data,
- to "generate a high-quality linked open dataset (LOD) describing live TV programming",
- to "combine this dataset with behavioral information to provide highly accurate market research information about viewing behavior", and
- to "employ the information gathered to build a recommendation service to provide real-time viewing recommendations."

There are six partners involved in the program:

- University of Zurich, represented by the "Dynamic and Distributed Information Systems Group". They are also the coordinator of the whole project.
- TU Dortmund University, represented by the "Artificial Intelligence Group".
- Vrije Universiteit Amsterdam, represented by the "Web & Media Group".
- *Zattoo* Europa AG
- *British Broadcasting Company* (BCC)
- *Rapid-I GmbH*, the developers of the RapidMiner suite for machine learning (see chapter 3.5.2).

The two IP-TV-Providers (*Zattoo* and *BBC*) deliver live video streams from selected broadcasting companies (also referred to as content providers). They furthermore gather anonymized data about consumers viewing behavior by recording user program switches. This whole data is then delivered to the three research institutions, where it is taken "as the input for a holistic live-stream data mining analysis". In addition to the video-

[3] http://vista-tv.eu/

and user-data the IP-TV-Providers provide the opportunity to receive EPG (Electronic Program Guide)-data for the video data as well.

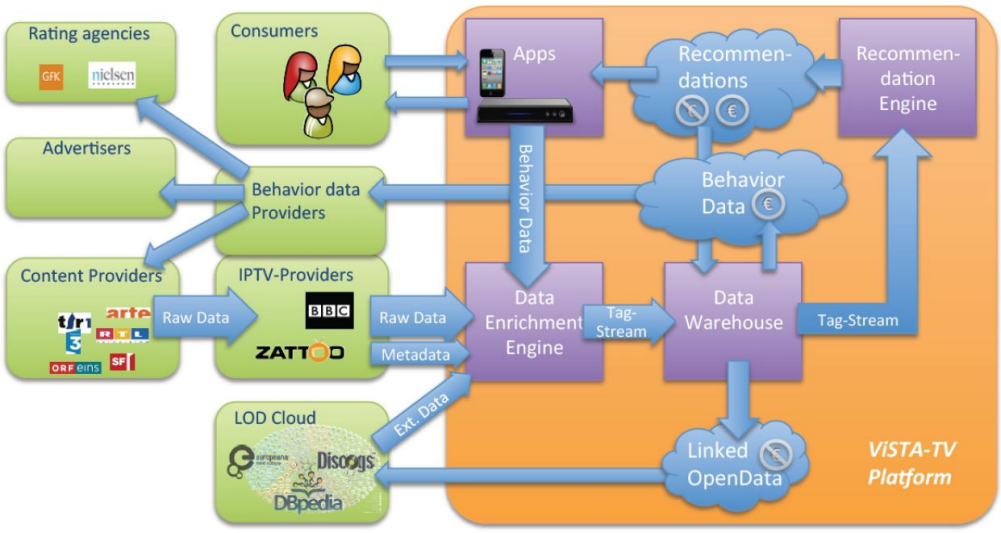


Figure 1.2.: Overview of the ViSTA-TV project

Afterwards the data is enriched with *internal* and *external* features. This is done by the module "Data Enrichment Engine". The term *internal* features refers to all features that are extracted from the provided data itself. Examples for this are any features gathered from the video data. In contrast to this, *external* features are all context information taken from third party content providers such as DBpedia[4] or Discogs[5]. This module is mainly driven by TU Dortmund University. Its output is an enriched feature stream, enabling the following modules to do further market analysis and create recommendations for users. This is done by the "Recommendation Engine" module. The "Data Warehouse" module is responsible for building up the infrastructure to store and deliver the enriched data.

Parts of the topic of this diploma thesis are supposed to be used in the "Data Enrichment" workpackage of the ViSTA-TV project.

1.2. Task of this thesis

Part of the ViSTA-TV project is to analyze, whether there are observable events within the video, audio or text streams that make people switch channels, and combine this with external features such as peoples' interests or peoples' activities on social media platforms. This requires the identification of video events. Identifying video events again can be seen as the problem of segmenting a video stream in meaningful parts and tagging these parts. This again requires the extraction and evaluation of features, that might

[4]http://de.dbpedia.org/
[5]http://www.discogs.com/

be useful in the context of video segmentation and tagging video segments. As we can assume that

1. the amount of possible features extractable from video data is almost infinite and
2. we do initially not know which features are useful

we start by extracting a maximized number of exemplary features. As analyzing the video stream and creating recommendations is supposed to happen in real-time, the amount of selected features must not be too large at the end. Especially complex features (in terms of time needed to calculate them) should be avoided to guarantee real-time behavior.

As a next step the extracted features have to be applied to real-world problems in order to evaluate, which subset of features is useful. Hereby the identification of features that appear to be useless is an important result as well, as these features do not have to be taken into account when later on video streams are analyzed in real-time. In this thesis two important video segmentation and tagging problems for IP-TV broadcasts have been considered as use cases (= applications):

1. The segmentation of videos by detecting cuts and gradual transitions, and
2. the tagging of news videos by distinguishing between anchorshots and news report shots.

There are of course other reasonable segmentation and tagging tasks as well, such as the identification of advertisement in movies and tagging the segments of a broadcasted movie with the label "movie" or "advertisement". But as the set of features, that have to be extracted to solve other tasks, will be similar, I decided to focus on the above mentioned use cases.

Furthermore a second use case has been built up. This one is closer related to surveillance applications than to IP-TV, as it is about supervising the coffee consumption for our office. The learning tasks for both use cases are defined in more detail later on.

Last but not least, we have to take into account that the feature extraction is supposed to be run in real-time on live video streams. Hence this thesis should demonstrate, how feature extraction in real-time can be performed and keep an eye on the runtime of segmentation and tagging algorithms.

1.3. Structure of this work

This work is structured in eight chapters. This *Introduction* chapter includes a motivation of the problem and an introduction of the context. The subsequent chapters are structured as follows:

Chapter 2 provides a literature survey of the field of video segmentation and tagging. Beside a definition of the terms segmentation and tagging, promising approaches for different segmentation tasks are introduced.

Chapter 3 gives a rough introduction to the field of machine learning. Alongside with some notation, machine learning tasks and algorithms, which are used in this thesis, get introduced. Furthermore evaluation methods for machine learning algorithms are presented. In addition the tools, especially RapidMiner and the streams framework, which play an important role for this thesis, get introduced.

Chapter 4 presents two use cases. Based on these use cases, I tested the above mentioned video segmentation and tagging techniques. The learning tasks to be solved in this thesis are defined and quality measurements are established in order to be able to evaluate the yielded results.

Chapter 5 gives an overview of the datasets that were created for the use cases.

Chapter 6 provides the reader with an overview of possibly extractable features from video data. The feature extraction has been performed both in a batch mode, using RapidMiner, and online, using the streams framework.

Chapter 7 contains a documentation of the experiments ran on the previous presented datasets. All results are evaluated in regard to the quality measurements, which were defined in chapter 4.

Chapter 8 concludes this thesis by summarizing the yielded results and proposing topics for future research.

2

Video Segmentation and Tagging

Definition 1 *The term* ***Segmentation*** *denotes the "process of partitioning data into groups of potential subsets that share similar characteristics". [Li and Ngan, 2011]*

Having a look at Definition 1, it becomes obvious that Video Segmentation covers a wide range of different tasks and emerges in various application areas. Generally the term can refer to two major categories of segmentation tasks

1. the **spatial segmentation** of videos into objects, as well as
2. the **temporal segmentation** of video data into temporal sections.

(a) spatial segmentation

(b) temporal segmentation

Figure 2.1.: Results of spatial and temporal segmentation on the same video data. Spatial segmentations tries to identify objects with similar characteristics (in this example faces and persons), whereas temporal segmentation tries to identify temporal segments by identifying frames with similar characteristics (in this example separated by the dash).

Spatial segmentation denotes the segmentation of video content into objects, that share similar characteristics. This includes Object Recognition as well as Foreground and Background Identification and is used in various application areas.

Face Detection and **People Detection** refers to the task of localizing human faces. Application fields include motion tracking in fun applications like video game consoles (i.g. XBOX kinect) as well as security applications or the face priority autofocus capacity of digital cameras and web-cams [Rahman and Kehtarnavaz, 2008].

Face Recognition extends the face detection task by adding functionality to identify the detected person. It can be used for security applications like access control of buildings or to detect unusual events in a given environment by analyzing surveillance videos [Stringa and Regazzoni, 2000, Zhong et al., 2004].

License Plate Recognition is another popular subtask of object recognition. It is used in order to find stolen vehicles or collect tolls [Chang et al., 2004].

Environment Classification is used to detect and track objects like streets and obstacles (i.g. [Farabet et al., 2011]). This enable autonomous vehicles ([Turk et al., 1987], [Pomerleau, 1993]) or planes ([McGee et al., 2005]) to find their way in an unknown environment.

Foreground and Background Identification can, for example, be used to reconstruct 3D settings from images or video data [Vaiapury and Izquierdo, 2010].

We can imagine some of these approaches to be useful for the segmentation and tagging of video content. For example, by detecting faces in video data, we could recognize anchorshots or interviews in news shows. Furthermore face recognition might be useful to identify actors in movies and hence help to tag scenes. Thus, some of the above mentioned approaches will be picked up later (i.g. Face Detection in news videos, chapter 2.3.3). But the main task of this thesis is the temporal segmentation of video data in meaningful parts. Therefore I will focus on temporal video segmentation approaches in the further course of this chapter.

2.1. Temporal hierarchy of video content

"Recent advances in multimedia compression technology, coupled with the significant increase in computer performance and the growth of the Internet, have led to the widespread use and availability of digital videos" [Yuan et al., 2007]. In order to efficiently use the information given in video material on the web, it is necessary to efficiently index, browse and retrieve the video data. So far this is most often done manually, but as the amount of video data is increasing rapidly, automatic parsing and matching has become an important and quickly growing field of research.

When aiming on cutting a video into meaningful segments, we will find various useful levels of abstraction. This hierarchy differs from author to author, but most approaches will agree that television broadcasts consist of shows on the first level. These shows can

then be divided into segments with the same topic which may consist of multiple scenes. Scenes themselves might then be segmented into shots and on the lowest level shots consist of single frames (identical to single images). Figure 2.2 shows an example of this hierarchy based on a fictitious "ZDF Sportstudio" show.

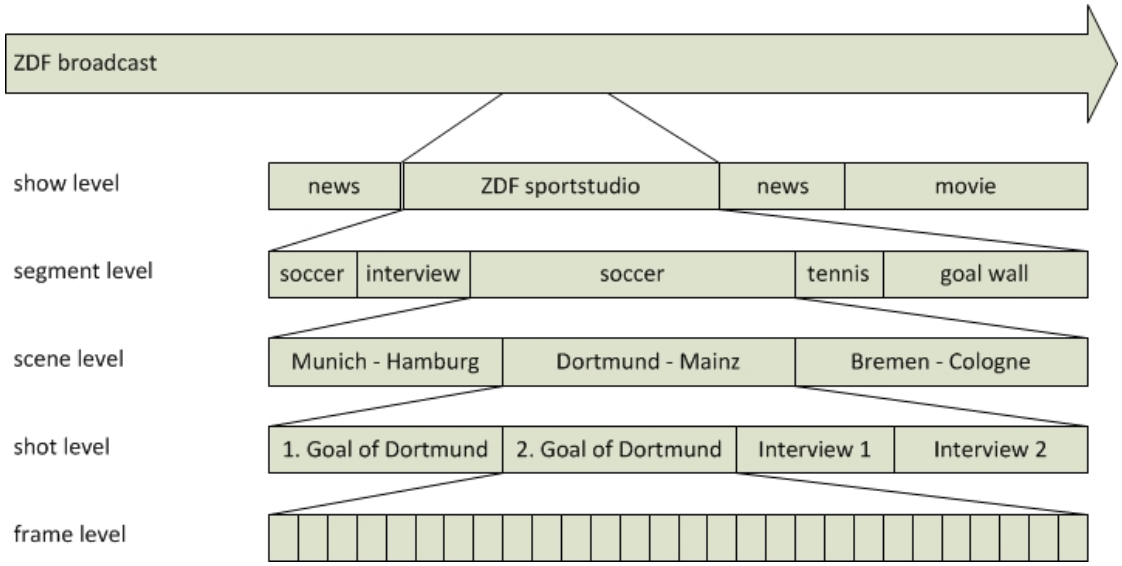


Figure 2.2.: Hierarchical temporal segmentation of a television broadcast.

When talking about video content broadcasted over television, the first level of this hierarchy (show level) is given by the available EPG data. Furthermore the decoded video stream consists of single frames, so that the lowest level of the hierarchy (frame level) is of course given as well. The task of segmenting a video into meaningful sections hence aims of the identification of shots, scenes and segments. This can be done top-down by trying to split a show into parts that are so diverse that it is unlike they belong together.Another option is going bottom-up by grouping frames together to receive a segmentation on the shot level and than grouping shots together to receive scenes or segments. As almost all existing approaches operate bottom-up, I am first going to focus on the task of shot boundary detection (section 2.2). In the following section 2.3, I will introduce approaches that allow to group the identified shots together.

2.2. Shot Boundary Detection

For any segmentation task on each level of the hierarchy, a segmentation of the video stream into constituent shots will be necessary. This learning task is equivalent to the task of finding shot boundaries in video sequences (shot boundary detection, also known as temporal video segmentation). As this is a basic task a lot of approaches for automatically recognizing and characterizing shot boundaries have been developed in the past 20 years.

Definition 2 (Shot (Traditional)) *Originally a shot is an image sequence consisting of one or more frames and recorded contiguously during a single operation of the camera.*

A shot is hence presenting a continuous action in time and space. (Definition is a combination of the definitions found in [Brunelli et al., 1996] and [Smoliar and Zhang, 1994])

This definition has been blurred due to the new possibilities digital videos offer. In news shows for example the anchorperson is sometimes talking about more than one topic within one operation of the camera. Figure 2.3 (b) shows a good example. Anyhow, in these cases the background changes and it will be helpful to identify this as different shots. Therefore it makes sense to slightly modify the definition of a shot and weaken it up. So my new definition is:

Definition 3 (Shot) *Shots are image sequences without any fundamental changes between the images within one shot. Especially all traditional shots are shots.*

I am aware of the fact that this definition is less precise than the traditional one and leaves room for interpretation. But it will turn out to be handier in real applications like shot boundary detection in news shows where the video-content is digitally edited before it gets broadcasted.

Basically there are two different types of transitions between two shots: hard cuts (also known as abrupt cuts or camera breaks) and more sophisticated gradual effects like fades, dissolves and wipes [Bordwell and Thompson, 2012].

(a) hard cut

(b) gradual transition

Figure 2.3.: Examples for transitions between two shots taken from the "Tagesschau" (German news program).

hard cut Shot A is directly followed by shot B. There is no kind of gradual effect between the last frame of A and the first frame of B. Figure 2.3 (a) shows an example for this.

fade The term fade describes the transition from a blank image into shot A (fade-in) or vice versa (fade-out). When shot A fades out into a monochrome image and afterwards shot B is faded in, the transition between shot A and shot B is called fade-out/in.

dissolve A dissolve overlaps shot A and shot B for a certain duration. While shot A gradually disappears, shot B gradually appears. Figure 2.3 (b) shows an example for this.

wipe A wipe is a transition where shot B reveals by travelling from one side of the frame to another or with a special shape. Famous examples for wipes are barn door wipes (where the wipe proceeds from two opposite edges of the image towards the center) or matrix wipes. Wipes are often used in presentations but they are very rare in TV broadcasts.

In their comparison paper from 1996 Boreczky and Rowe (University of California Berkeley) [Boreczky and Rowe, 1996] have shown that hard cuts are the most common type of transition. Gradual transitions are less frequent. Their results, based on 3.8 hours of TV broadcasts, are shown in table 2.1.

Video Type	# of frames	Cuts	Gradual Transitions
Television	133.204	831	42
News	81.595	293	99
Movies	142.507	564	95
Commercials	51.733	755	254
Miscellaneous	10.706	64	16
Total	419.745	2507	506

Table 2.1.: Frequency of hard cuts versus gradual transitions in different video types. [Boreczky and Rowe, 1996]

Literature

Beside [Boreczky and Rowe, 1996], serveral more detailed surveys exist, comparing further shot boundary detection approaches and/or working on different data sets. Some were already written in the 1990's ([Aigrain and Joly, 1994], [Ahanger and Little, 1996], [Patel and Sethi, 1996] [Lienhart, 1999] or [Browne et al., 1999]), others are newer like [Gargi et al., 2000], [Koprinska and Carrato, 2001], [Hanjalic, 2002], [Lefevre et al., 2003], [Cotsaces et al., 2006] or [Yuan et al., 2007]. Furthermore, since 2001, the shot boundary detection problem has been included as a task in the Text REtrieval Conference series (TREC), sponsored by the National Institute of Standards and Technology (NIST). Since 2003, one of the workshops on this conference (TRECVID[1]) only handles video data [Smeaton et al., 2006]. This also involves a good video dataset to benchmark new approaches.

The amount of literature existing in this field reflects the number of different approaches for shot boundary detection. Of course, all methods base on the quantification of the difference between consecutive frames in a video sequence and most of them have in common, that features based on the image colors are chosen to distinguish between different shots. Swain and Ballard [Swain and Ballard, 1991] have proven, that "color has excellent discrimination power in image retrieval systems. It is very rare that two images

[1] http://trecvid.nist.gov/

of totally different objects will have similar colors" [Zhang et al., 1995c]. Nevertheless the different approaches vary a lot. Mostly only two consecutive frames are considered. This works well for hard cuts, but is often not enough to detect gradual transitions. Hence we are first focusing on the detection of hard cuts. In a later subsection I address the problem problem of detecting gradual transitions.

2.2.1. Detection of hard cuts

As hard cuts are the most common type of transition between two consecutive frames, a lot of different approaches have been researched to detect them. Most of them quantify the difference of two images based on the color, but some also focus on edge detection or motion vector comparison. In this paragraph, some selected approaches are described.

Pair-wise pixel difference

A quite simple way to compare two images in order to figure out, whether they are significantly different or not, is to determine, how many pixels have changed. These approaches are most often used on monochromatic images (see chapter 6.2.1), but can of course be transfered to color images easily. One of the first approaches was published by Nagasaka et al. in 1991 [Nagasaka and Tanaka, 1992]. They simply add up the differences of intensity of all pixels (x, y), with $x \in X, y \in Y$, where X and Y denote the width and height of all frames, over two successive frames F_i and F_{i+1}. This sum is then compared to a given threshold T and a shot boundary is declared, when T is exceeded.

$$(\sum_{x \in X} \sum_{y \in Y} |F_i(x, y) - F_{i+1}(x, y)|) > T$$

Using this metric, a small amount of pixels that are dramatically changing can already cause an exceedance of the threshold. Zhang et al. [Zhang et al., 1993] therefore developed an approach with two thresholds. One to judge a pixel (x, y) as changed or not (PC=Pixel Changed) and one to detect the shot boundaries themself. A pixel is declared as changed in the $i + 1$th frame, if the difference of its grayscale values between frame F_i and frame F_{i+1} exceeds a given threshold t.

$$PC(x, y) = \begin{cases} 1 & \text{if } |F_i(x, y) - F_{i+1}(x, y)| > t \\ 0 & \text{otherwise} \end{cases}$$

As soon as at least T percent of the total number of pixels changed between frame F_i and frame F_{i+1}, the frames are declared to be right before respectively right behind a shot boundary.

$$\frac{\sum_{x,y} PC(x, y)}{max(x) + max(y)} * 100 > T$$

Unfortunately this easy metric does not work well in many settings, as it is very sensitive towards object and camera movements. Given the camera is panning, each pixel (x_1,y_1) in frame F_{i+1} will be identically to a pixel (x_2,y_2) close by in frame F_i. But it might

have a huge difference compared to the same pixel (x_1,y_1) in frame F_i. Therefore "a large number of pixels will be judged as changed even if the pan entails a shift of only a few pixels" [Zhang et al., 1993]. Similar problems occure, when an object in the foreground is moving in front of a fixed background. Hence the number of detected shot boundaries is usually too high, as camera and/or object motions are mistakenly classified as shot boundaries (false positives). To handle this problem, it has proven to be useful to apply a smoothing filter to the image before calculating the pair-wise pixel difference. For example, Zhang et al. tried to smooth the images applying a 3×3 unweighted image smoothing filter before computing the pair-wise pixel comparison [Zhang et al., 1993]. For further information on image filtering, please see chapter 6.2.1.

Statistical difference

Instead of comparing all pixels pair-wise, there are also approaches that compute statistical measures over adjacent frames and compare these statistical measures. Simple statistical measures are i.g. the average value of all pixels after transforming the image to grayscale. Other approaches propose more complex statistics. For example, Kasturi and Jain [Kasturi and Jain, 1991] proposed an approach, computing the mean value (m) and the standard deviation (S) of the gray levels of two successive frames and have a measure called likelihood ratio based on these two values. If the likelihood ratio exceeds a given threshold T, a shot boundary is declared.

$$\frac{[\frac{S_i+S_{i+1}}{2} + (\frac{m_i-m_{i+1}}{2})^2]^2}{S_i * S_{i+1}} > T$$

This approach is quite resistant against noise, but unfortunately the computation of the statistical measures is slow due to the complexity of the statistical formulas.

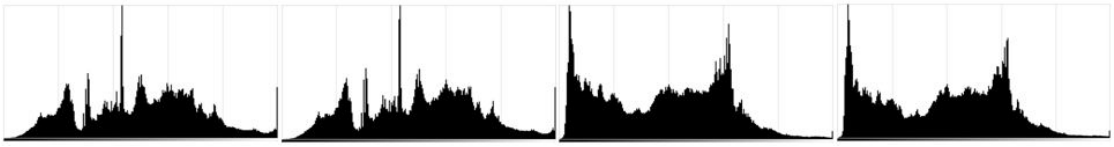
Histogram difference

Other approaches do not rely on statistical measures but on color histograms. Color histograms are a discretized representation of the color distribution of an image. For each discretized color value (=color bin) it is counted, how many pixels fall into that bin. The evaluation of color histograms is described in chapter 6.2.1. Based on the histograms of two adjacent frames, a shot boundary is declared, when the bin-wise difference of the two histograms exceeds a given threshold T. The histograms can be either calculated on grayscale images [Tonomura and Abe, 1989] or the color image itself. Nagasaka and Tanaka [Nagasaka and Tanaka, 1992], for example, use color histograms with 64 bins (2 bits for each RGB color component [Lefevre et al., 2003]).

$$\sum_{j=1}^{G} |H_i(j) - H_{i+1}(j)| > T$$

In this formula $H_i(j)$ denotes the percentage of pixels in bin j, with $1 \leq j \leq G$, for the ith frame. G is the total number of bins.

(a) Four successive frames taken from a "Tagesschau" news show.



(b) Corresponding color histograms

Figure 2.4.: Difference of gray-level histograms around a shot boundary

Basing on this easy color histogram difference, other authors tried to improve the detection rate by modifying the formula. Dailianas et al. [Dailianas et al., 1995], for example, have developed a method that "tries to amplify the differences of two frames" by squareing them. Shots are declared, when

$$\sum_{j=1}^{G} \frac{(H_i(j) - H_{i+1}(j))^2}{max(H_i(j), H_{i+1}(j))} > T$$

This method is known as color histogram difference based on χ^2-tests and goes back on Nagasaka and Tanaka [Nagasaka and Tanaka, 1992]. "Since the color histogram does not incorporate the spatial distribution information of various colors, it is more invariant to local or small global movements than pixel-based methods." [Yuan et al., 2007]

Block-based difference

Instead of applying metrics to entire frames, we can also think of comparing smaller blocks (also referred to as regions or grids) of successive frames. Therefore each frame is segmented into blocks. Afterwards the shot boundary detection gets either based on all blocks or just a selection of blocks. Kasturi et al. [Kasturi and Jain, 1991] for example cut each frame into a fixed number of regions. Then the likelihood ratio for each corresponding block is computed and a shot boundary is declared if the likelihood ratio of more than T frames exceeds a given threshold t. Nagasaka and Tanaka [Nagasaka and Tanaka, 1992] have developed a similar approach. They divide each frame into 4×4 grids and compare the corresponding regions of successive frames. Hence they get 16 difference values. Instead of using all of them, they base the shot boundary detection on the eight grids that have the lowest difference values. Hereby their approach gets robust towards noise, but is prone to panning.

Net comparison by Xiong et al. [Xiong et al., 1995] uses a block-based approach in order to speed the shot boundary detection up. They break a frame into base windows and only take some of these base windows into account in order to detect a shot boundary. By only

looking at some blocks (= a selection of all pixels), they outperform other approaches with regard to computational time.

Compression differences

As the video data is present in a digital format, other authors base their approaches on features, that can directly be extracted from the encoding. In order to understand these methods, we first have to look at the encoding formats.

H.264/MPEG4-AVC (Advanced Video Coding) is today's most common standard to encode digital videos. It is also used by *Zattoo* and thus of interest for the ViSTA-TV project. Encoded videos consist of three types of frames: I-, P- and B-frames [Richardson, 2003], [Strutz and Strutz, 2009].

I-frames, also called "Intra-coded pictures", are video frames that have been encoded without taking into account any other frame of the video. Hence I-frames can be seen as stand-alone pictures and are encoded like still JPEG images.

P-frames ("Predicted Pictures") and **B-frames** ("Bi-predictive pictures") are only holding part of the image information and require the prior decoding of other pictures in order to be decoded. Without further distinguishing between P- and B-frames, we can simplifyingly assume that those frames basically represent the difference between the current frame and the frame(s) before (or even after) it. If, for example, an object moves in front of an unchanging background, only the object's movement gets encoded. Everything else (in this setting the background) is implicitly assumed as unchanged. Thus space is saved, which lowers the required broadband when transmitting movies over the Internet. Figure 2.5 shows an example for this. One technique of encoding only the difference between two images is called motion compensation encoding. The encoding of motion is done by discrete cosine transformation (DCT).

(a) Frame 1 (b) Frame 2 (c) Difference image

Figure 2.5.: Pictures (a) and (b) show two successive frames from a video sequence. Frame (a) is transmitted as an I-frame. Instead of transmitting frame (b), only the difference image (c), holding everything that changes from (a) to (b) is transmitted as a P-frame. In the difference image (c), mid-grey stands for "nothing has changed", whereas light and dark greys represent positive and negative differences respectively. [Richardson, 2003]

All pictures between one I-frame and the next, including the first I-frame, are called a group-of-pictures (GOP).

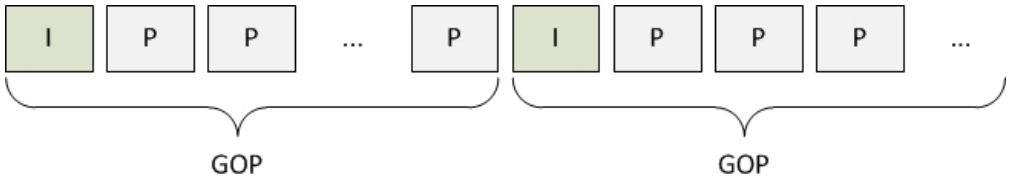


Figure 2.6.: Group-of-pictures (GOP)

As the similarity in successive frames is quite large within one shot, encoders tend to prefer P- and B-frames to encode the video frame. On the other hand, consecutive frames with hardly no similarity are encoded as I-frames. This is especially the case for frames right after a hard cut. Therefore the occurrence of I-frames is a good indicator for the presence of a shot boundary. Furthermore the P- and B-frames hold information about the motion of objects from one frame to the next. This information is captured in so called motion vectors (MVs). The comparison of the length, direction and coherence of motion vectors over several images can hence be used for shot boundary detection, as well.
Shot boundary detection on compressed video data has, for example, been done by Arman et al. [Arman et al., 1993]. They are using the discrete co-sinus transformations (DCT)-features in an encoded MPEG-stream to find shots. Lee et al. [Lee et al., 2000] use "direct edge information extraction from MPEG video data". Zhang et al. [Zhang et al., 1995a] detect shots based on the pair-wise difference of DCT coefficients gained from MPEG compressed video data.

Edge tracking

Instead of detecting shot transitions on the images themselves, Zabih et al. have proposed an algorithm based on transformed frames [Zabih et al., 1995]. They apply a border detection algorithm (see chapter 6.2.1) on each incoming frame first. Then they count the number of border pixels that do not correspond to any border pixel in the previous frame (window size = 2). This is identical to the number of border pixels, which is further than t pixels away from any border in the previous frame. The higher this amount of changed borders is, the more likely it is that a shot boundary exists. The comparison to some of the above mentioned approaches has shown, "that the simple histogram feature usually is able to achieve a satisfactory result while some complicated features such as edge tracking can not outperform the simple feature." [Yuan et al., 2007]. Lienhard [Lienhart, 1999] has furthermore pointed out that the edge tracking method is computationally much more expensive than the simple color histogram methods. Hence I decided to not further test this approach.

2.2.2. Detection of gradual transitions

Comparing gradual transitions to cuts, we find that the difference values of two successive frames are significantly smaller for gradual transitions. Hence the above mentioned

approaches for the detection of hard cuts will most likely fail on gradual transitions. Intuitively we would try to lower the threshold, but unfortunately it turns out that in many cases the difference values resulting from a gradual transitions will be smaller than the difference values of successive frames resulting from camera movements, zooming or panning. Because of that we need techniques that enable us to distinguish between gradual transistions and special camera effects.

Twin comparison

The twin comparison approach has been developed by Zhang, Kankanhalli, and Smoliar [Zhang et al., 1993]. Instead of using just one threshold, this approach bases on a lower threshold T_l and a higher threshold T_h. Whenever the difference of one frame towards its' previous frame exceeds the lower threshold T_l, the frame gets declared as the potential start of a gradual transition F_s. As long as the difference value of two successive frames does not drop under the threshold T_l, these frames are compared to the potential starting frame F_s, resulting in an accumulated difference value. If this accumulated difference value exceeds the higher threshold T_h, which is by the way also used to detect hard cuts, the end of a gradual transition F_e is detected. If the difference value between two consecutive frames drops below T_l before the accumulated difference value exceeds T_h, the potential start point F_s is dropped and no shot boundary is declared. The approach is illustrated in figure 2.7, taken from the comparison paper of Ahanger [Ahanger and Little, 1996].

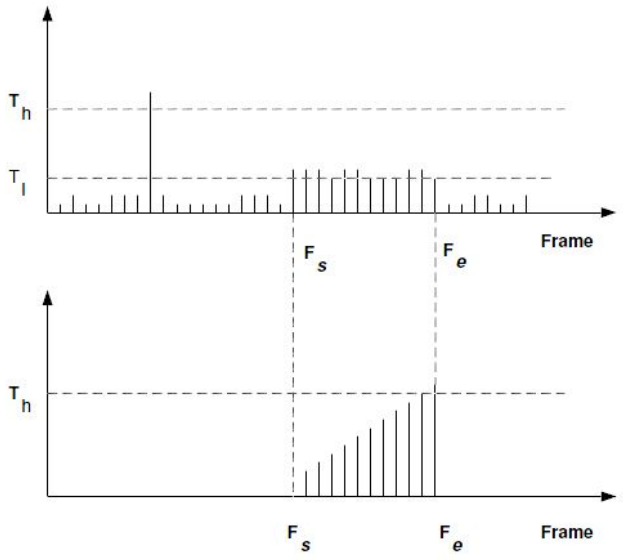


Figure 2.7.: Illustration of the twin comparison approach. Figure taken from [Ahanger and Little, 1996].

Unfortunately the approach is not robust towards special camera effects. Hence we still have the problem to distinguish between gradual transitions and special camera effects. This can be done by using motion vectors.

Motion vectors

In order to distinguish between gradual transitions and special camera effects, Zhang et al. [Zhang et al., 1993] proposed a method to infer the reason for high differences between consecutive frames. Their approach bases on motion vectors. By tracking the movement of single pixels between consecutive frames, we gain motion vectors. Tracking pixels can be done by searching for pixels with the same color in two consecutive frames. In case there is more than one pixel having the same color in the second image, the spatially closest one is assumes to be the one we are looking for. For special effects like pans, zoom-ins and zoom-outs these motion vectors generate typical fields, shown in figure 2.8.

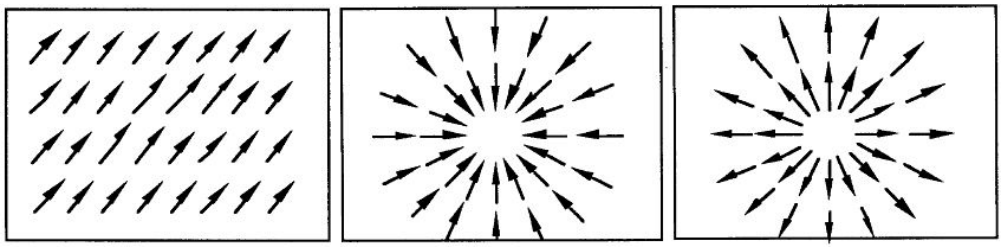


Figure 2.8.: Illustration of motion vectors due to camera panning, zoom-out and zoom-in. Figure taken from [Zhang et al., 1993].

If all computed motion vectors point in the same direction, the difference of the images most likely occurred due to a pan of the camera. In case all motion vectors are directed from the image border towards a single center of focus, we detect a zoom-out. Analog, if all motion vectors are directed from a single center of focus towards the image border, we detect a zoom-in. Of course we should ignore small disparities that might occur due to the movement of objects or noise. If the extracted motion vectors do not match any of the three patterns, we assume that a gradual transition is the reason for the difference of the frames.

Instead of computing the motion vector information on the decoded image stream, these information can also be gained from MPEG encoded video data directly. "However, the block matching performed as part of MPEG encoding selects vectors based on compression efficiency and thus often selects inappropriate vectors for image processing purposes." [Boreczky and Rowe, 1996]

2.3. Segment Detection

As one hour of video data mostly consists out of hundreds of shots, the shot level is surely not the ideal level for indexing video data as indexes would still be way too large to give a good overview of the present video content. Therefore "it is necessary to find more macroscopic time objects, for instance larger sequences constituting a narrative unit or sharing the same setting." [Aigrain et al., 1997]. To solve this learning task, almost all approaches rely on the detection of shot boundaries. Hence the detection of shot

boundaries, as described in section 2.2 "is an important step towards the automated extraction of meaningful video segments" [Brunelli et al., 1996]. The next thing to do is grouping together the shots into meaningful scenes.

Performing this macro-segmentation on random video content turns out to be very difficult. The problem is that "automatic construction of content-based indices for video source material requires general semantic interpretation of both images and their accompanying sounds" [Zhang et al., 1995b], something which "is beyond the capabilities of the current technologies of machine vision and audio signal analysis" [Zhang et al., 1995b]. Hence "researchers have been focusing on special applications utilizing domain knowledge" [Gao and Tang, 2002] to develop solutions for special tasks. Focusing on video content that we have a priori knowledge of, makes the segmentation task significantly easier. Popular application fields include news broadcasts ([Zhang et al., 1995b], [De Santo et al., 2007], [Bertini et al., 2001]), sports programs (Goalgle [Snoek and Worring, 2003b], [Snoek and Worring, 2003a], [Xu et al., 2001]), or commercial advertisements. For these types of video content, scientists have found promising approaches to automatically segment video data and extract useful annotations.

A brief introduction of methods, applicable for random video content, is given in the following section 2.3.1. As I am not going to perform the segmentation tasks on random video content but on news videos only, section 2.3.2 provides you with an overview of existing approaches for the segmentation of news videos.

2.3.1. General-purpose methods for segment detection

Most of the approaches for shot boundary detection are based on the idea that those frames, which belong together, are similar and therefore share a set of characteristics. Unfortunately this approach can not just be transfered to the task of identifying segments, as frames in one segment are possibly very diverse. A good example for this is a shot reverse shot setting. This is a film technique that is often used in interviews or movies when two characters, which face each other, are talking to each other. An example of an interview making use of this technique is shown in figure 2.9.
Neither the colors, nor the background, the amount of faces nor any other edge or motion vector comparison will be able to classify these shots as one segment. Nevertheless there

Figure 2.9.: Example for a shot reverse shot setting, taken from an interview of Jürgen Klopp, the coach of Borussia Dortmund, broadcasted in the "ZDF sportstudio" of September 2nd, 2012.

are approaches that are able to discover such segments, without having further domain knowledge. Two of them are detailed in this section.

Rule-based segment detection

This approach goes back on Aigrain, Joly, and Longueville [Aigrain et al., 1997]. They claim that microscopical changes in movies and shows are emphasized by using stylistic devices. Examples for such stylistic devices are the use of gradual transition effects, modification of the audio track, or variations in the editing rhythm. Hence we can take the presence of these stylistic devices as clues, in order to solve the video segmentation task. In the above mentioned paper the authors define a total of ten rules that enable us to group shots together in order to get a segmentation of a video on the scene level. I pick three types of rules to demonstrate, what they look like. For more details please see the original work.

Transition effect rules Possible transitions in videos are cuts (C) and gradual transitions (GT). Hence the transitions occurring in a video file can be seen as a word over the alphabet $\{C; GT\}$. Transition effect rules base on the recognition of sub-words, which follow a certain pattern. For example if the sub-word $C^i GT^j C^k$ with $i, k > 2, j = 1$ is found, we can assume that there is a segment boundary right before the beginning of the gradual transition.

Shot repetition rules Especially within a shot reverse shot setting, the same shot is repeated within a distance of just a couple of shots in between. The shot repetition rule aims on detecting shots like that by comparing the first frame after a transition with representative frames of the last three shots. Beside interviews or movies using the shot reverse shot technique, this rule will turn out to be useful to handle overexposed frames due to photographic flashlights. This will be further described in chapter 7. On the other hand the application of the rule can result in the false grouping of shots. For example anchorshots in news shows occur repeatedly. Nevertheless they belong to different news stories and should hence not be grouped together.

Editing rhythm similarity rule Editing rhythm corresponds to the amount of transitions occurring in a video. The more transitions we have, the shorter the duration of each single shot is and the faster the movie appears to the viewer. Hence the shot duration in action scenes will be rather short in comparison to dialogs. This is utilized by the editing rhythm rule. As soon as a shot is three times longer or four times than the average of the last ten shots, the shot is likely to be important and can hence be seen as a segment boundary.

Audio-based segment detection

Segment boundaries are usually indicated by boundaries of the video- and the audio-stream. Hence taking the audio data into account can be very helpful for the detection of segments. Commercial spots, for example, most often consist of many shots and transitions, but the audio stream is probably uninterrupted. Furthermore constant background

music or an ongoing comment from a speaker can also indicate that two shots belong to the same segment.

Nevertheless the audio data solely is almost never sufficient to detect segments. Hence more approaches that take into account audio data base on both, the audio- and video-stream. To mention one approach, Aigrain et al. [Aigrain et al., 1997] have included some rules in their rule-based segment detection approach that base on the audio stream. They claim a segment boundary, if there has been no sound for at least 30 seconds and then sound starts, whilst a transition in the video stream was detected at the same point. Obviously this rule is very strict and does not apply often.

As I focus on the extraction of video features, I do not want to go into further detaisl at this point. Further information about audio analysis is provided in chapter 2.5.1.

2.3.2. Segmentation of news shows

News shows are an excellent use case for the segmentation and tagging of video data, as they have a well known temporal structure. The typical temporal structure of a news show can be found in figure 2.10. Following this simplifying schema, a news show consists of a set of anchorshots, each one followed by related news shots.
As one anchorshot along with the following news shots typically form a news story, they can be seen as a scene. Hence scenes in news shows, in contrast to scenes in other shows, are often referred to with the special name news storys.

Definition 4 (News story) *A news story is "a segment of a news broadcast with a coherent news focus which contains at least two independent, declarative clauses." [Kraaij et al., 2005]*

The detection and annotation of news stories used to be done manually, which is quiet costly but necessary in order to keep track of the data in big news video archives [Merlino et al., 1997]. Therefore "documentalists (...) manually assign a limited number of keywords to the video content" [Snoek and Worring, 2005]. As broadcast agencies as well as private consumers and governmental institutions depend on this quick access to news archives, a lot of effort has been put in the automation of this process. Based on the detection of shot boundaries, "news can be finally segmented into news stories; each story is obtained by simply linking a given anchorshot with all the following news report shots until another anchorshot or the news end occurs." [Santo et al., 2004]. Hence

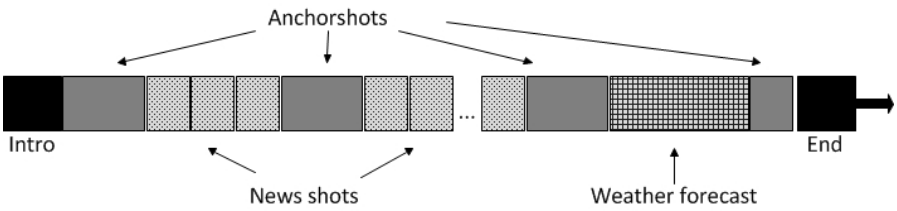


Figure 2.10.: The typical temporal structure of a news show as described in [Zhang et al., 1995b].

anchorshot detection is the key challenge in order to analyze, segment, and browse news video data.

2.3.3. Anchorshot Detection

First of all we have to define the term anchorshot.

Definition 5 (Anchorshot) *Each shot within a news show showing one or more anchorperson(s) is called anchorshot.*

Anchorshots can vary in the number of anchorpersons (mostly one or two), the absence or presence of a news icon and title in the background, or a labeling bar. If no news icon or title is present, the anchorperson will usually be in the center. As soon as the anchorshot shows one anchorperson and a news icon, the anchorpersons' position will be on the right side and the news icon will be on the left or vice versa. In case of two or more anchorpersons, the anchorshot typically has no news icon. This holds for most news shows world-wide.

Figure 2.11.: Example for an anchorshot.

As anchorshots are widely used to cut news shows into meaningful segments, the detection of anchorshots has been of great interest in the scientific community. Thus, shots in news shows get typically classified into anchorshots and news report shots. First a given video data gets segmented into shots. Afterwards one or more frames from each shot get selected as key frames for this shot. Subsequently anchorshots get classified by either matching the current key frame against a model for anchorshots (supervised) or by trying to recognize similar shots that occur repeatedly during the news show (unsupervised). In the following sections, some approaches get presented and their advantages and disadvantages will be discussed.

Model matching

All approaches presented in this section try to classify anchorshot by taking benefit from the a priori knowledge we have about anchorshots. Swanberg et al. [Swanberg et al., 1993] propose a system that matches shots against a unique anchorshot model. As soon as the current shot fulfills the characteristics of this model, it is labeled as an anchorshot. Their method might work well as long as there are no variations between the anchorshots, but Zhang et al. [Zhang et al., 1995b] have tested this model matching approach on news videos broadcasted by the *Singapore Broadcasting Corporation* (SBC) and received poor results due to the fact, that these news shows cover anchorshots with different spatial structures. Hence, they have developed a system that matches each shot against a set of models. Their matching is based on the pair-wise pixel difference and on color histograms. Unfortunately, their approach still depends on the predefined models and does not work on any news show. Hanjalic et al. [Hanjalic et al., 1999] have tried to overcome

this problem by introducing a semi-automatic approach that finds its own template for anchorshots. Based on key frames for each shot, they compute the dissimilarity of each shot with all other shots. Then they look for a shot, which best matches other shots throughout the show and assume that shot to be a good template for an anchorshot. The disadvantages of this approach are quiet obvious: Again this approach only holds a single model for anchorshots and will not work well in settings where anchorshots vary a lot. Especially when different anchorshots do not share the same background, which might be due to different camera angles, their matching does not work well. Furthermore the finding of the anchorshot template requires to compare each shot to all other shots within the show. Hence, the approach can hardly be adapted to a real-time environment, as each shot could only be compared to the shots before. Thus the approach would have to be approximated to make it available in a stream environment.

Face detection

Since the lightning, studio design or dominant colors of anchorshots can vary between different news shows or over time, Avrithis et al. [Avrithis et al., 2000] and others have based their anchorshot detection algorithms on the only characteristic that can not vary: the presence of an anchorperson. They identify anchorshots by recognizing the anchorperson's face by a face detection module. Their module recognizes faces by color matching and shape recognition. In contrast to other approaches, they do not try to identify people. They are only interested in the presence or absence of a person's face somewhere in the foreground. Because of this interviewers, reporters or politicians at press conferences might incorrectly be labeled as anchorpersons and various news shots will consequently be classified as anchorshots. Günsel's approach [Gunsel et al., 1996] overcomes these problems by also taking into account the position and size of the detected faces.

Frame similarity

Similar to Hanjalic et al. [Hanjalic et al., 1999], other authors try to find anchorshots by looking for shots throughout a video, which share a set of visual features. These approaches use the characteristic that are "extremely similar among themselves, and also frequent compared to other speech/report shots." [Ide et al., 1998] Therefore they can be found by clustering all shots of a news show and assuming that the "largest and most dense cluster would be the anchorshots". As this technique is unsupervised and does not need any anchorshot model, it promises to perform better in unknown environments, like on news shows we have never seen before. On the other hand it is unlikely, that they outperform approaches that use more a priori assumptions or have anchorshot models for the specific news show available.

One approach using cluster analysis was presented by Gao et al. [Gao and Tang, 2002] in 2002. They apply a graph-theoretical cluster analysis (GTC Analysis) on key-frames from news shows. By constructing a minimum spanning tree over vertices that represent those key-frames and cutting edges that exceed a given threshold, clusters are gained. All clusters with more than two nodes in them are then taken as potential anchorshots. In

order to decrease the misclassification rate, further assumptions are made. For example it is assumed, that anchorshots are at least two seconds long. Furthermore similar frames are only taken into account, if they appear at least 10 seconds apart from each other. Nevertheless misclassifications occur due to anchorshots that only appear once per show and similar news shots that appear more than once throughout a news show. Bertini et al.[Bertini et al., 2001] assume that anchorshots differ from all other shots in regard to motion. As neither the anchorperson nor the background or the camera are moving, their classification approach bases on the presence or absence of motion vectors. But as digital animations are getting more and more usual in nowadays television shows, their assumption might already be wrong in many cases. Furthermore zooming is also a widely common camera technique in some news shows and could cause motion in anchorshots.

Unfortunately these approaches again assume that we have seen the whole news show at the time we want to label the anchorshots. This assumption is fine in case we want to organize and structure a news archive, but it is not given in a real-time environment. Hence the approaches will not be applicable in the ViSTA-TV project.

Multi-expert systems

As none of the above mentioned approaches provides fully satisfactory performance on arbitrary news videos, it might be useful to combine some of the above mentioned techniques in a multi-expert system (MES). Such a system uses different classifiers for anchorshots to classify the incoming data. Afterwards the results of all classifiers are combined by taking an average or majority voting. A promising approach has been developed by De Santo, Percannella, Sansone, and Vento [Santo et al., 2004], using three different unsupervised classifiers.

2.4. Tagging the segments

Beside the segmentation of the video data we are moreover interested in assigning tags to the video material.

Definition 6 (Tag) *A tag is a term or keyword assigned to a piece of data, describing the data or including valuable meta data.*

The segmentation of video data itself already provides a set of tags for videos. For example we can use the shot boundary detection to directly assign the keyword "shot boundary", "first frame of a shot", or "representative frame" to single shots. Furthermore the segmentation of news videos allows us to label frames as "anchorshot frames", or assigning keywords like "news story 1" to the data. But in order to better understand the user behavior, it might be necessary to add way more tags to the incoming video data stream. For this thesis I only focused on the above mentioned tags. Nevertheless this section gives a brief overview of further tags, that might be useful in the future.

Advertisement Another interesting tag might be the segmentation of broadcasted movies into "advertise" and "movie" respectively. This recognition of advertisement has been done extensively and approaches vary a lot. Some again utilize

a-priori knowledge, other approaches base on databases that contain known commercial spots and compare the video stream to the stored spots. The database can then be expanded by adding segments between known commercial spots and assuming, they are advertisement as well. A-priori knowledge includes that advertisement spots are full of motion, ending with a rather still shot, presenting the product. Furthermore the audio signal is mostly tuned up during commercials and in many cases two commercial spots are separated by some monochrome frames. Lienhart, Kuhmünch, and Effelsberg [Lienhart et al., 1997a] for example have applied both approaches on German television data, gaining very good results.

News story names Anchorshots usually contain a textual label, which briefly summarizes what the following news story is about. This captions can be read out using Optical Character Recognition (OCR) software [Sato et al., 1999]. Approaches include locating text areas, applying filters to sharpen the text and keyword detection.

Events Beside its application in the tagging of news videos, OCR can be of benefit in other settings as well. By reading out the overlaid scores in sport broadcasts, Babaguchi et al. [Babaguchi et al., 2004] were for example able to detect highlights in football games. They used their method to automatically create short video clips, containing all highlights of a game.

Actors By using face detection methods, some approaches try to identify certain actors in video data. As actors usually act in costumes and with makeup on, this turns out to be very complex by just matching the detected faces against a database of actors' faces. Some approaches recognize the reoccurrence of certain characters within the same movie [Lienhart et al., 1997b]. By taking into account EPG-data as well, this can help to identify the main characters and provides useful tags for video data as well.

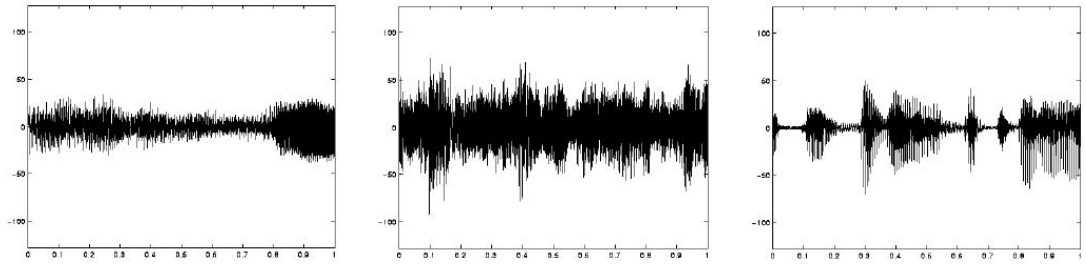
2.5. Further related work

In many applications, video data comes along with audio data. Hence it suggests itself to take into account audio data to improve the quality of the segmentation of the video data. The following section gives an overview of the existing approaches that handle audio data.

2.5.1. Audio analyze

Daxenberger [Daxenberger, 2007] gives a good overview of the state of the art in the field of audio segmentation. The described approaches do no take into account any video data and have mainly been designed to segment audio data like music files. Daxenberger has furthermore developed an application called *Segfried*, which is able to automatically segment audio data using different segmentation techniques. The implemented segmentation algorithms include a segmentation by Support Vector Machines (SVMs), and a metric pair-wise constraints k-Means Clustering (MPCK-Means).

Mierswa and Morik [Mierswa and Morik, 2005] address the problem of automatically classifying audio data. They extract features from music files and predict the genre, in which the music file falls, based on the extracted features. Lui et al.[Liu et al., 1998] use audio data to classify television broadcasts. They observed that low-level audio features already have a good differentiation power for classifying "commercials", "basketball games", "football games", "news", and "weather forecasts". This is already indicated when looking at the waveforms of different audio data. Figure 2.12 shows three examples.



(a) commercials (b) basketball (c) news

Figure 2.12.: Waveforms for different audio data, corresponding to video data. Figures taken from [Liu et al., 1998].

2.5.2. Multi-modal analyze

Beside the aforementioned rule-based segmentation algorithm by Aigrain et al. [Aigrain et al., 1997], other scientists have also developed approaches for the segmentation of video data that can be seen as multi-modal approaches, taking into account video and audio data. Sundaram and Chang [Sundaram and Chang, 2000], for example, segment the audio and video data into scenes separately. Afterwards they determine, at which shot boundaries as well the audio scene detection as the video scene segmentation approach claim scene boundaries. Only those that were detected on as well the video as the audio data are taken into account.

Snoek, and Worring [Snoek and Worring, 2005] and Wang et al. [Wang et al., 2000] have written good comparisons of further approaches. In the further course of this work, I will focus on features extracted from video data. As video data does not come along with audio data in any setting (e.g. surveillance cameras and especially the "coffee" dataset I am using later on), I am not going to take into account audio features at all.

3

Machine Learning

The acquisition of new knowledge based on experience is generally called learning. Aside from humans and animals, some machines possess the ability to learn. These machines take empirical data as their input and try to identify underlying patterns or generalizations on this data. The gained knowledge is than used to make predictions on new data and helps to make intelligent decisions based on data that has never been seen before. This process is called machine learning and is an important branch of artificial intelligence.

Definition 7 (Machine Learning) *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [Mitchell, 1997]*

Figure 3.1.: Example of handwritten digits

To illustrate this definition, we introduce a common engineering example for machine learning, that is also included in [Hastie et al., 2001]: The handwritten digit recognition.

In order to enable machines to sort letters by their ZIP codes, they have to be able to recognize handwritten digits on the envelopes. The problem is, that handwritings differ a lot and therefore the same digit can look totally different on different envelopes. Figure 3.1 shows examples for this. Humans have learned to generalize and are able to identify various handwritings, even if they have never seen the handwriting before. The challenge is to teach this ability to a machine. Hence, the task T is to classify handwritten digits. The program bases on a set of examples of handwritten digits. This set forms the experience E. An obvious performance measure P is given by the recognition rate, which is the percentage of correct recognized digits. As it is impossible to memorize all possible appearances for each digit, the recognition has to be based on general characteristics, that have been observed on the experience E.

This entire chapter bases on the standard references about machine learning. Namely I want to mention two books: "The Elements of Statistical Learning" by Hastie, Tibshirani and Friedman [Hastie et al., 2001] and "Machine Learning" by Mitchell [Mitchell, 1997].

3.1. Notation

The input of a machine learning algorithm is some experience E. The experience is represented as set of examples also referred to as training set X_{train}. This set contains N examples $\vec{x}^{(i)}, i \in \{1, N\}$

$$X_{train} = \{\vec{x}^{(1)}, ..., \vec{x}^{(N)}\}$$

Each example $\vec{x}^{(i)} \in X_{train}$ is a p-dimensional vector. Here, p is the number of features $A_j, j \in \{1, p\}$ that form one example. The value of the j-th feature in the i-th example is referred to as $x_j^{(i)}$. Without loss of generality we can assume that the number of features p and their order are constant throughout the training set X_{train}. Features can either be numerical, nominal or binominal.

Numerical features have values ranging over $\mathbb{N}$ or $\mathbb{R}$. They are the most common feature type, since a numerical representation supports statistical analysis best. Values can be either continuous or discrete. Most physical measurements are examples of continuous data, like i.g. temperature, length or speed. In contrast, the age of a person is a discrete numerical feature, as the number of possible outcomes is finite. Continuous data can be transfered to discrete data by summarizing values over a certain number of value ranges. This process is known as discretization.

Nominal features have values ranging over an unordered and discrete set of possible values. A good example, taken from the popular iris flower discrimination dataset[1], is for example the species of an iris flower, that gets distinguished between three possible types: $\{iris\ setosa, iris\ virginica, iris\ versicolor\}$.

Binominal features are nominal features that only have two possible outcomes. Most often these two outcomes are $\{true, false\}$.

[1] http://archive.ics.uci.edu/ml/datasets/Iris

A machine learning algorithm takes its input X_{train} and looks for some model. A common type of a model is a function $\hat{f}$ that maps elements from the input space X to elements from the output space Y:

$$\hat{f} : X \to Y$$

When this model gets applied to an example from the test data $x^{(i)} \in X_{test}$, it produces an output $\hat{y} \in Y$. Simplifying we can assume, that each $\hat{y}$ is a label.

According to [Hastie et al., 2001] the field of machine learning splits into two major subtasks: supervised and unsupervised learning. The distinctions results from the presence or absence of labels in the training data X_{train}.

3.2. Supervised Learning

Supervised Learning is the task of inferring a model $\hat{f}$ from labeled training data. The input data set X_{train} for supervised learning algorithms again consists out of N examples, each represented by a p-dimensional vector. One feature of each example is designated as the true label for that example. Therefore we can also view X_{train} as a set of pairs $(\vec{x}, y)$, where y is the label.

$$X_{train} = \{(\vec{x}^{(1)}, y^{(1)}), ..., (\vec{x}^{(N)}, y^{(N)})\} \subset X \times Y$$

Based on this input data, the machine learning algorithm infers a function that predicts labels $\hat{y} \in Y$ for unlabeled test data X_{test}. In case the number of possible outputs is discrete (often $y \in \{+1, -1\}$), the model is called a classifier. If the predicted output is continuous ($Y \subseteq \mathbb{R}$), the model is a regression function. Respectively the learning tasks are called **classification** and **regression**.

Definition 8 (Classification) *Classification is the machine learning task of inferring a function $\hat{f} : X \to Y$ from nominally labeled training data X_{train}. The model is chosen with respect to a quality criterion g that must be optimized.*

Definition 9 (Regression) *Regression is the machine learning task of inferring a function $\hat{f} : X \to Y$ from numerically labeled training data X_{train}. The model is chosen with respect to a quality criterion g that must be optimized. In many cases the label is only binominal.*

The above mentioned introducing example of recognizing handwritten digits on mailing envelopes is a typical example for a classification problem. The input of the learning algorithm consists of labeled examples of handwritten digits. The inferred model later predicts nominal labels $\hat{y} \in \{0, 1, 2, ..., 9\}$ for unlabeled examples.

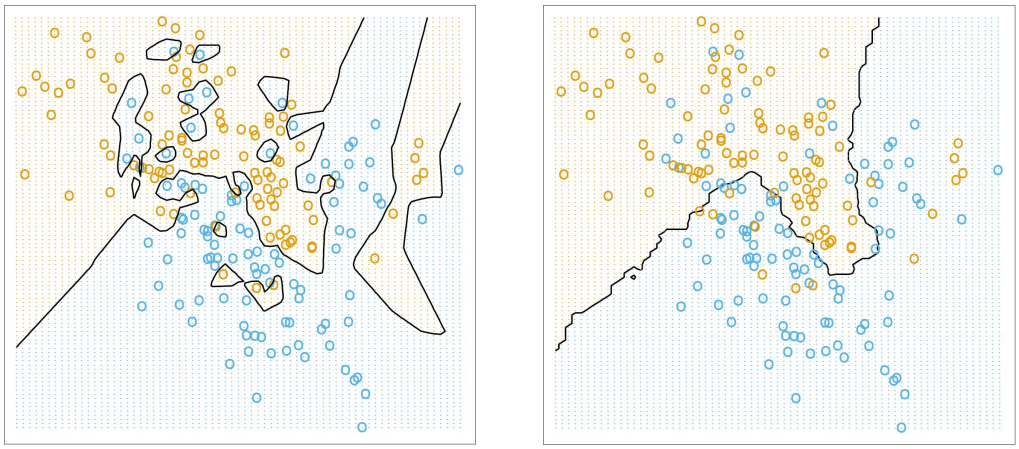
Examples for supervised learning algorithms are linear regression, k-nearest-neighbor algorithms (kNN), support vector machines (SVMs) or decision tree learners, just to mention a few. In following kNN and decision trees are explained in further detail, as they are used in chapter 7.

3.2.1. kNN

The k-nearest-neighbor algorithm (kNN) is a very simple and intuitive classification algorithm. At training time, the algorithm simply stores all labeled training data $X_{train} \subset X \times Y$ in its model. A new, unlabeled example $x \in X$ is then classified by simply searching for the k nearest neighbor examples in the training set X_{train} and predicting the most common label among these neighbors as the label $\hat{y}$ for x. Hence $\hat{f}$ can formally be defined as follows

$$\hat{f}(x) = \arg\max_{y^{(i)} \in Y} \left((x^{(i)}, y^{(i)}) \in N_k(x, X_{train}) \right)$$

where $N_k(x, X_{train})$ denotes set of the k nearest neighbors of x in the training data set X_{train}. Hence $|N_k(x, X_{train})| = k$. Figure 3.2 shows examples for k=1 and k=15 and a binominal label.



(a) k = 1 (b) k = 15

Figure 3.2.: Examples for the classification of two-dimensional data by k-nearest-neighbor classifiers with k=1 and k=15. The two classes are represented by the color. The blue shaded region denotes that part of the input space classified as BLUE. The orange shaded region stands for the ORANGE class respectively. Figures taken from [Hastie et al., 2001].

The term "near" of course depends on the chosen distance measure d. Possible distance measures are all functions $d : X \times X \to \mathbb{R}$, that fulfill the following requirements:

1. Distance is never negative: $d(x^{(1)}, x^{(2)}) \geq 0$ [2]
2. The distance of each object to itself is zero: $d(x^{(1)}, x^{(1)}) = 0$
3. The distance of two objects is symmetric: $d(x^{(1)}, x^{(2)}) = d(x^{(2)}, x^{(1)})$
4. The direct way from one object to the other is always shorter than making a detour over any third object: $d(x_1, x_2) \leq d(x^{(1)}, x^{(3)}) + d(x^{(3)}, x^{(2)}), x^{(3)} \in X \setminus \{x^{(1)}, x^{(2)}\}$

[2] In some cases, distance measures with negative distances are also allowed. These are not taken into account here.

In literature the term distance measure equals the term dissimilar measure, as the idea behind this measure is, that objects with a huge distance between each other are not similar to each other. Popular examples for distance measures are for example the Euclidean distance, given by

$$d(x^{(1)}, x^{(2)}) = \sqrt{(x_1^{(1)} - x_1^{(2)})^2 + (x_2^{(1)} - x_2^{(2)})^2 + ... + (x_j^{(1)} - x_j^{(2)})^2}$$

or the Manhattan distance, defined as

$$d(x^{(1)}, x^{(2)}) = |(x_1^{(1)} - x_1^{(2)})| + |(x_2^{(1)} - x_2^{(2)})| + ... + |(x_j^{(1)} - x_j^{(2)})|$$

In Figure 3.2 the Euclidean distance was chosen.

3.2.2. Decision trees

Decision trees are an intuitive way to represent a prediction model $\hat{f}$ and hence decision tree learners are a commonly used method in machine learning. "A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute [= feature], each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label." [Han and Kamber, 2006]. An example for such a decision tree is shown in figure 3.3.

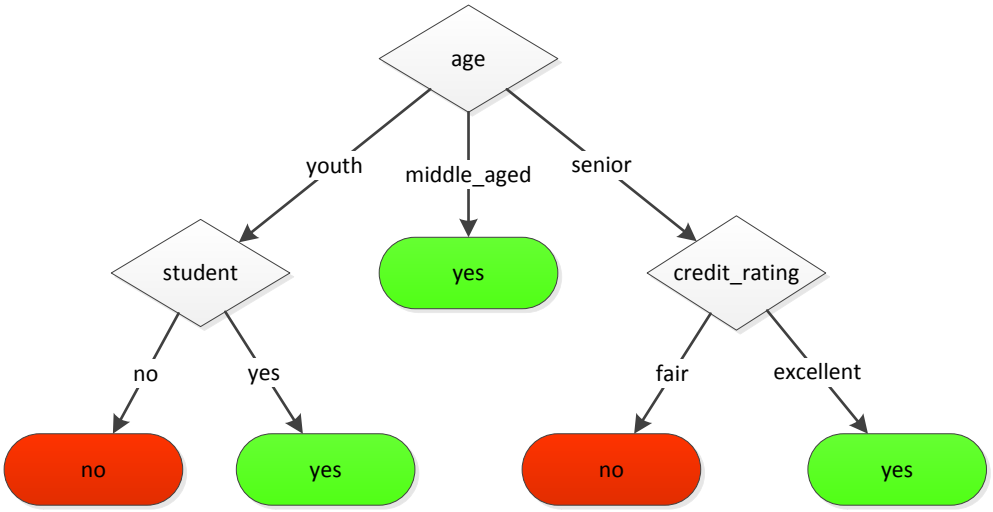


Figure 3.3.: A decision tree to predict, whether a customer at an electronic store is likely to buy a computer or not. Example taken from [Han and Kamber, 2006].

For this thesis, it is sufficient to focus on the top-down induction of decision trees (TDIDT), as top-down approaches are by far the most common method to infer a decision tree from given training examples X_{train}. TDIDT approaches base on two

tree learning algorithms, developed independently between the late 1970s and early 1980s: Iterative Dichotomiser (ID3) by J. Ross Quinlan [Quinlan, 1986] and Classification and Regression Trees (CART) by L. Breiman, J. Fiedman, R. Olshen, and C. Stone [Breiman et al., 1984]. The idea is quite simple and straightforward: At each step, the set of examples is divided into smaller subsets by splitting it by the values (or value ranges) of one feature. The feature is hereby selected in a manner that it optimally discriminates the set according to the classes. Hence, an ideal feature would split the example set into subsets, that are pure (i.e. after the partition all examples in each subset belong to the same class). As most likely no feature will split the example set into pure subsets, different measures for the quality of a split have been proposed.

Information Gain The information gain criterion uses the entropy of a set of examples to find the optimal feature for splitting the set into subsets. The entropy of a set S measures the impurity of S and is defined as

$$E(S) = -\sum_{j=1}^{k} P(C_j) log_2(P(C_j))$$

where $P(C_j), j \in \{1, ..., k\}$ denotes the probability that an example in S belongs to class C_j. We now assume that we split S into v subsets $\{S_1, S_2, ..., S_v\}$ based on the value of a feature A in S. Afterwards we calculate, how helpful this splitting would be by calculating the accumulated weighted entropy over all resulting subsets $S_m, m \in \{1, ..., v\}$.

$$E_A(S) = \sum_{m=1}^{v} \frac{|S_m|}{|S|} \times E(S_m)$$

The information gain $Gain(A)$ of feature A is then defined as

$$Gain(A) = E(S) - E_A(S)$$

and the feature with the highest information gain $Gain(A)$ is chosen to split the example set S. For nominal features the number v of subsets is hereby usually the number of possible outcomes of the feature. For numerical features, any v can be chosen and v value ranges for the numerical features can be defined. The information gain criterion was introduced in ID3.

Gain Ratio By choosing the information gain criterion, features with a large number of different values are more likely to be chosen than other features. Especially in a setting, where one feature holds an unique value for each example (i.e. an ID), this feature would be selected, as it splits the example set in a way, that each subset is pure. Hence it makes sense to penalize features by their number of possible values v. This is done by the gain ratio criterion. The gain ratio $GainRation(A)$ of a feature A is defined as

$$GainRatio(A) = \frac{Gain(A)}{SE_A(S)}$$

with

$$SE_A(S) = -\sum_{m=1}^{v} \frac{|S_m|}{|S|} \times log_2(\frac{|S_m|}{|S|})$$

Gini Index The Gini Index was introduced in CART and considers binary splits only. Again the impurity of a set of examples S before and after a potential split is measured. The feature A, that reduces the impurity most, is then selected as the splitting feature. The Gini Index $Gini(S)$ is defined as

$$Gini(S) = 1 - \sum_{j=1}^{k} P(C_j)^2$$

where $P(C_j), j \in \{1, ..., k\}$ again denotes the probability that an example in S belongs to class C_j. Assume feature A is chosen as the splitting feature and S gets split into subsets S_1 and S_2 the impurity after the split is given by

$$Gini_A(S) = \frac{|S_1|}{|S|} Gini(S_1) + \frac{|S_2|}{|S|} Gini(S_2)$$

Based on these values, the split is carried out for the feature A that reduced the impurity of S best.

$$\Delta Gini(A) = Gini(S) - Gini_A(S)$$

For nominal features with v possible outcomes, the optimal partitioning of S is determined by testing all potential combinations of binary subsets.

The recursive splitting of the example set S is performed until a certain stop condition is fulfilled. This could be that the size of the tree has reached a certain level, the purity of the leaves exceeds a given threshold or there is no feature left over that really helps to reduce the impurity further.

3.2.3. Evaluation Methods

To evaluate the performance of machine learning algorithms, it is necessary to define a set of performance measurements. This section gives a short overview of the performance measurements that are used in chapter 7.

First of all we have to define the expression performance. Performance can be viewed from different perspectives: It can either relate to the time it takes to build a model, to how long the classification of an training example takes or to the scalability and storage usage of an algorithm. All these performance criteria are reasonable, but will not be detailed in this thesis. When talking about performance, we mainly focus on accuracy of an algorithm. The model $\hat{f}$, produced by a learning algorithm, is said to have a perfect accuracy, if the predicted output $\hat{y^{(i)}}$ for each example $\vec{x}^{(i)} \in X_{train}$ equals the true label $y^{(i)}$. Obviously this could easily be achieved by a learning algorithm by simply memorizing all data given in X_{train} and not generalizing at all. Hence the model complexity would be very high, but the prediction error on the training data would be zero. Nevertheless such an algorithm would surely perform poor on unseen test data. This phenomenon is known as bias-variance-tradeoff and shown in figure 3.4. In order to handle this problem, we have to decrease the complexity of the model and test it on different data than the data it was trained on. Hence it is necessary to split the given

data into test- and training-data. This can be done by simply putting some data aside (holdout validation) or, which is way more common, performing cross-validation.

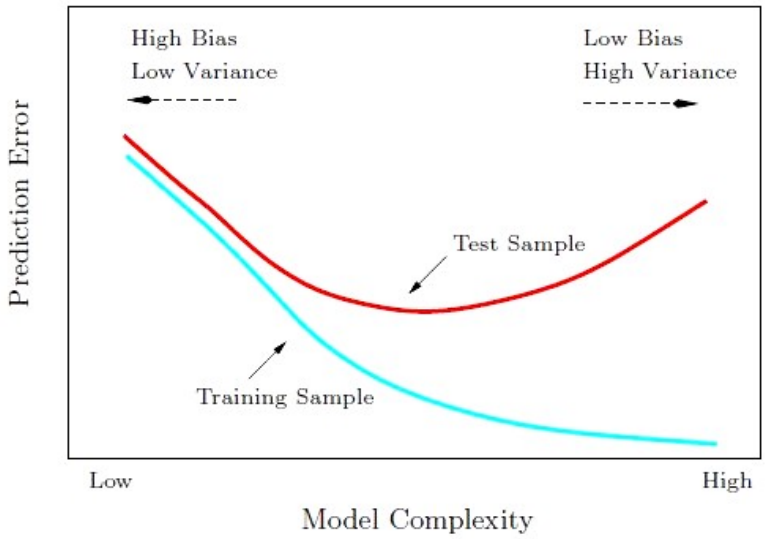


Figure 3.4.: The bias-variance-tradeoff: Test and training error as a function of model complexity. Taken from [Hastie et al., 2001].

K-fold cross-validation In K-fold cross-validation (X-validation) the given training data $X_{train} = \{(\vec{x}^{(1)}, y^{(1)}), ..., (\vec{x}^{(N)}, y^{(N)})\}$ is split randomly into K roughly equal-sized subsets, each containing N/K examples. One of these subsets is than retained as test data, whereas the other $K-1$ subsets are used to train the model. This process is repeated for each of the K subsets (K-folds).

The advantage of cross-validation is, that all data gets used for training as well as testing, which is important, as usually the amount of labeled data is limited and hence we do not have enough data to perform holdout validation. According to [Hastie et al., 2001], typical choices of K are K=5 or K=10. If K equals the number of examples in X_{train} (K=N), only one example is left out in each fold. Hence this variation of K-fold cross-validation is called leave-one-out cross validation (LOOCV).

The estimated prediction error of a model can now be defined as the average difference of the prediction $\hat{y}$ and the true label y. For regression problems where $y, \hat{y} \in \mathbb{R}$, this is quite easy. For classification problems we introduce the confusion matrix as a visualization of the performance of an algorithm.

Definition 10 (Confusion Matrix) *The confusion matrix is a table, that allows visualization of the performance of a classification algorithm. The columns of the table represent the predicted labels $\hat{y}$, the rows represent the actual labels y.*

Going back to the example of handwritten digit recognition, the confusion matrix of a learning algorithm could be given by table 3.1.

	Prediction: 0	Prediction: 1	...	Prediction 9	Total
Label: 0	94	3	...	7	130
Label: 1	2	81	...	2	98
...	...	...	...	...	...
Label: 9	3	0	...	87	108
Total	122	98	...	131	1000

Table 3.1.: Possible confusion matrix for the handwritten digit recognition problem.

As we can see, we had a total of 1000 examples. 130 were labeled as 0, 98 were labeled as 1 et cetera. The inferred model predicted the label $\hat{0}$ for 122 examples. 94 out of that 122 were labeled correctly, the other 28 examples were labeled wrong. The same goes for all other classes. Thus the accuracy of the model is given by summing up all elements on the main diagonal (=correct classified examples) and dividing them by the total number of examples. By the way, the error is then given by summing up all entries of the confusion matrix except those on the main diagonal.

Definition 11 (Accuracy) *Accuracy is the proportion of elements, which really belong to the class they were predicted to belong to right predictions, based on the whole population.*

$$Accuracy = \frac{|Correctly\ predicted\ items|}{|All\ items|} \tag{3.1}$$

In many real-world classification problems the number of classes is binary. Thus these classification tasks are called binary classification tasks. Examples are the classification of e-mail into spam and no spam, the classification of patients in those having a certain disease or not, and the classification of news shots into anchorshots and news report shots (see chapter 2.3.3).

In such a setting the resulting confusion matrix (see table 3.2) basically consists out of four entries:

true positives (TP), counting all examples, that belong to the class we are looking for and our model predicts this correctly,

true negatives (TN), holding all examples, that do not belong to the class we are looking for and get labeled as such,

false negatives (FN), counting all items, that belong to the class we are looking but our model claims they do not

false positives (FP) holding those items, that do not belong do the class we are looking for but our model incorrectly claims them to do.

	Prediction: true	Prediction: false	Total
Label: true	TP	FN	...
Label: false	FP	TN	...
Total	...	...	...

Table 3.2.: Confusion matrix for a binary classification task

Obviously the number of correctly labeled examples is given by summing up the true positives and true negatives, whilst all examples belonging to the false negatives or false positives are misclassified examples.

$$Correct = TP + TN$$

$$Missed = FN + FP$$

Based on that, the accuracy for binary classification tasks is simply defined as

$$Accuracy = \frac{Correct}{Correct + Missed} = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.2}$$

Hence an accuracy of 100% is perfect, as this means that all predicted values are the same as the true values. For binary classification task we furthermore define precision and recall as appropriate evaluation criteria. These two values are commonly used.

Definition 12 (Precision) *Precision is the percentage of retrieved items that are desired items.*

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

Definition 13 (Recall) *Recall is the percentage of desired items that are retrieved.*

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

3.3. Unsupervised Learning

In contrast to supervised learning, unsupervised learning methods handle unlabeled data. These algorithms try to find underlying patterns in the input data X_{train}.

The task of recognizing handwritten digits on mailing envelopes, for example, can be transfered into an unsupervised learning task by modifying the input data. Therefor we remove the labels $y^{(i)}$ of all examples $(x^{(i)}, y^{(i)}) \in X \times Y$. The learning task then is to find similar patterns in the input data. We specify that we are searching for $k = 10$ clusters, and try to assign one cluster $C_j, j \in \{0, 1, 2, ..., k-1\}$ to each $x^{(i)}$. Given the clustering algorithms works well, the output will consist of labeled examples $(x^{(i)}, \hat{y}^{(i)})$, where each $x^{(i)}$ is assigned to one cluster. Unfortunately we do not have any information about

which cluster represents which digit. For example all handwritten digits representing a four could be enclosed in cluster C_7.

Examples for unsupervised learning algorithms are association rule learners and clustering. As clustering is used in chapter 7, it is described in more detail in the following subsection.

3.3.1. Cluster Analysis

Cluster Analysis refers to the machine learning task of grouping "a collection of objects into subsets or 'clusters', such that those within each cluster are more closely related to one another than objects assigned to different clusters." [Hastie et al., 2001] Two objects are supposed two be closely related (also called similar), when the distance between them is small. Hence the similarity of two examples is inverse to their distance. Simplifying we assume that

$$sim\left((x^{(1)}, x^{(2)})\right) = 1 - d(x^{(1)}, x^{(2)})$$

Figure 3.5 shows an example for a possible clustering of two-dimensional numerical data. The Euclidean distance is chosen as the distance measure and the resulting clusters are indicated by the color of the examples.

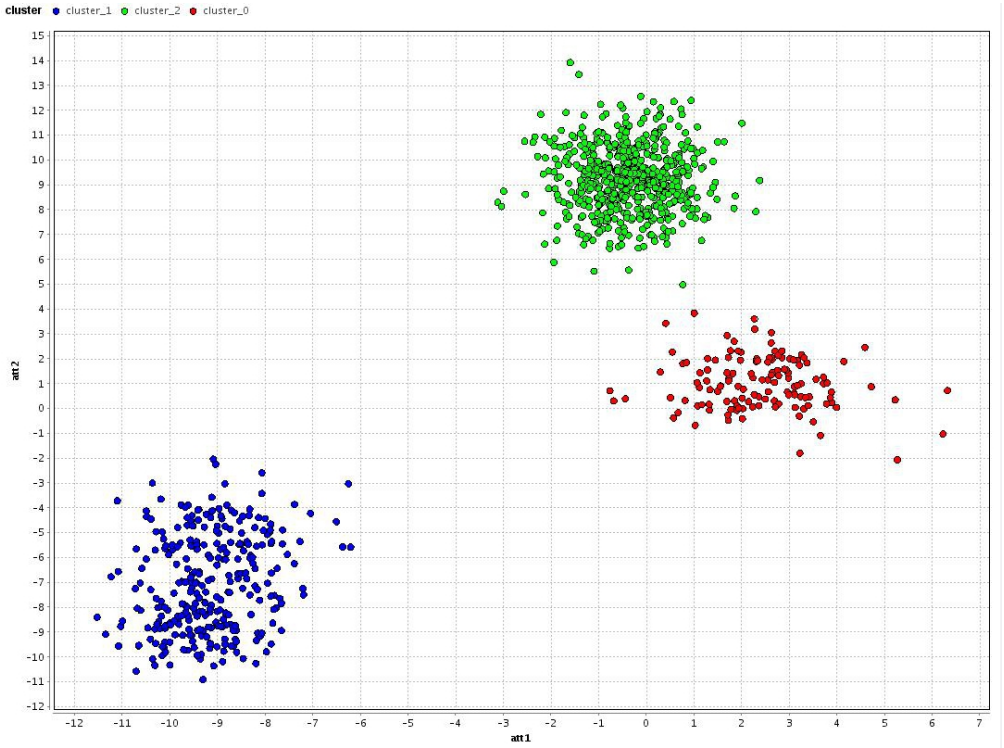


Figure 3.5.: Example for the result of a cluster analysis. Randomly generated two-dimensional data produced by using the 'Generate Data'-Operator of RapidMiner. Cluster Analysis is performed by the 'Clustering'-Operator using k-Means with k=3.

Formally, the task of a cluster analysis algorithm can be specified as a multi-objective optimization problem. On the one hand, the within cluster distance $W(\zeta)$ must be minimized, whilst on the other hand the between cluster distance $B(\zeta)$ must be maximized.

$$W(\zeta) = \frac{1}{2} \sum_{i=1}^{k} \left(\sum_{\vec{x}^{(a)} \in X | \zeta(\vec{x}^{(a)}) = C_i} \left(\sum_{\vec{x}^{(b)} \in X | \zeta(\vec{x}^{(b)}) = C_i} \left(d(\vec{x}^{(a)}, \vec{x}^{(b)}) \right) \right) \right)$$

$$B(\zeta) = \frac{1}{2} \sum_{i=1}^{k} \left(\sum_{\vec{x}^{(a)} \in X | \zeta(\vec{x}^{(a)}) = C_i} \left(\sum_{\vec{x}^{(b)} \in X | \zeta(\vec{x}^{(b)}) \neq C_i} \left(d(\vec{x}^{(a)}, \vec{x}^{(b)}) \right) \right) \right)$$

Hence we can define the term clustering as follows:

Definition 14 (Cluster Analysis) *Clustering, or Cluster Analysis, is the machine learning task of inferring a mapping $\zeta : X \rightarrow C$ that maps unlabeled data $X = \{\vec{x}^{(1)}, ..., \vec{x}^{(N)}\}$ to k clusters $C = \{C_1, C_2, ..., C_k\}$, in a way that each example $\vec{x}^{(i)}$ gets assigned to exactly one cluster, under the condition that $W(\zeta)$ is minimized, whilst $B(\zeta)$ is maximized.*

There are various algorithms that perform cluster analysis, differing a lot: Some of them allow objects to belong to more than one cluster at the same time (no strict partitioning), some do not group the objects into concrete clusters but rather evaluate probabilities for each pair of examples and clusters, saying how likely it is, that the example belongs to the cluster. As the topic is really broad, I do not take these approaches into account, but focus on easy clustering algorithms like k-Means ([MacQueen, 1967]).

k-Means

k-Means is an algorithm for cluster analysis. It partitions the incoming unlabeled data $X = \{\vec{x}^{(1)}, ..., \vec{x}^{(N)}\}$ into k clusters $C = \{C_1, C_2, ..., C_k\}$. Each of the incoming examples $\vec{x}^{(i)}$ gets assigned to exactly one cluster C_j. The clusters are represented by their centroids. Centroids are calculated by averaging the values of the corresponding features for all points in the cluster. Usually the centroid will be an imaginary point, that is not included in the data X.

The k-Means algorithm works iterative. Initially, k centroids are randomly chosen. Then each example is assigned to the cluster, it is closest to, by calculating the distance of the example to all centroids. When all examples are assigned to an cluster, the centroids for all clusters are updated. This procedure is repeated continuously, until none of the centroids moves any longer. The resulting centroids are then assumed to be good centroids to cluster the data.

3.4. Learning on Streams

All the above mentioned machine learning algorithms are designed to work on a fixed amount of data. This data is assumed to be stored in files or data bases and all data can be accessed at any time. This, for example, enables a decision tree learner, as described in section 3.2.2, to evaluate, which attribute splits the data best. For many years this traditional batch data processing has been sufficient. But as the amount of data rapidly increases, more and more data gets generated continuously, and consumers are interested in reacting to data drifts in real-time, these traditional approaches reach their limits. Hence learning on streams has become an upcoming part of machine learning in the last years. Examples for such environments include sensor networks or web log analysis and computer network traffic supervision for security reasons [Gaber et al., 2005].

The limitations given by operating on streaming data lead to the following requirements for streaming algorithms [Bockermann and Blom, 2012b]:

C1 It is necessary to continuously process single items or small batches of data,

C2 the algorithm uses only a single pass over the data,

C3 the algorithm may consume only limited resources (memory, time) , and

C4 the algorithm provides anytime services, which means that models and statistics have to be deliverable at any time.

In recent years, many approaches have been developed, solving most machine learning task on data streams. These algorithms include the training of classifiers on streams [Domingos and Hulten, 2000], stream clustering algorithms [O'Callaghan et al., 2002] [Aggarwal et al., 2003] (example given Birch [Zhang et al., 1996] or D-Stream [Chen and Tu, 2007]), quantile computation [Arasu and Manku, 2004], and approximate or lossy counting algorithms on streams (example given Lossy Counting [Manku and Motwani, 2002], Count(Min) Sketch [Charikar et al., 2004]), just to mention a very few. For further literature and explanations to the above mentioned algorithms I recommend the final report of the Projektgruppe 542, that took place at TU Dortmund University in 2010 [Balke et al., 2010].

The video data I am coping with in this thesis is provided in a streaming manner as well. The video data stream is infinite and the amount of data makes it impossible to store everything. Hence we do not have random access to the data and algorithms can only use a single pass over it. On these grounds it suggested itself to view IP-TV video streams as streaming data. Nevertheless only a very few of the video segmentation and tagging approaches described in chapter 2 make use of streaming algorithms. The reason is that there is most often no need for online learning. For tasks like cut detection or anchorshot detection, it is sufficient to build models offline and only apply them on the incoming video stream. Hence we are working in a batch learning but stream application environment and do not focus on the aspect of stream mining in too much detail.

Nevertheless some approaches developed for learning on streams could be useful for video segmentation and tagging as well. This section gives a short overview of machine learning approaches, which are applicable on streaming data.

3.4.1. Concept Drift Detection

In many applications, it is likely that the incoming data changes over time. As an example, we can think of an easy setting, where a regression learner tries to predict the amount of time needed to heat a pot of water based on the amount of water. After having seen several examples, the prediction will probably have a good accuracy. But as the initial temperature of the water from the tap might change seasonally and the power of the water boiler might decrease over time, the prediction model gets worse and worse. Hence it will be necessary to adjust the model to the altering setting. On the other hand we do not want to adjust our model to outliers. Hence, "instead of treating all training examples equally, a concept drift aware system must decide to what extent some particular set of examples still represents the current concept. After all, a recent concept drift might have made the examples less relevant or even obsolete for classifier induction" [Dries and Rückert, 2009].

Definition 15 (Concept Drift) *The term* ***concept drift*** *refers to the change of a statistical property of a label, which shall be predicted by an inferred model, causing the model to become less accurate over time.*

By detecting concept drifts and automatically adjusting the model, concept drift aware learning algorithms can improve the accuracy of a classification. As the underlying models in the setting of video data also change from time to time, it could be useful to adjust the models by using concept drift aware classifiers. A good example is the detection of anchorshots. The studio setting or the background might change from time to time, but the changes are mostly not significant. Nevertheless, a model based on the last anchorshots seen, will outperform a model that takes into account all data to the same extent.

3.4.2. Stream Clustering

Some approaches for unsupervised anchorshot detection based on the idea that the most dense cluster(s) of shots are probably anchorshots. These approaches have been developed for classifying batch data, but they could of course be adapted to streaming data by replacing the cluster algorithms with stream clustering methods. Examples for stream clustering methods are BIRCH [Zhang et al., 1996], STREAM [O'Callaghan et al., 2002], or D-Stream [Chen and Tu, 2007].

3.5. Tools and Frameworks

Although solving problems with machine learning approaches is a creative process, its' structure is often similar. This chapter gives an overview of the tools and frameworks I used to solve the machine learning task for this thesis. It starts with the CRoss Industry

Standard Process for Data Mining (CRISP-DM), which addresses this fact by defining a process model for carrying out data mining projects [Shearer, 2000]. Afterwards two software frameworks for solving machine learning tasks are presented: RapidMiner and the streams framework.

3.5.1. CRISP-DM

CRISP-DM was designed in 1996 by a European Union project led by four companies: *SPSS* (an *IBM* company), *Teradata*, *Daimler AG* and *OHRA*, a dutch insurance company. Regarding to CRISP-DM a data mining process consists of six phases, that are shown in figure 3.6:

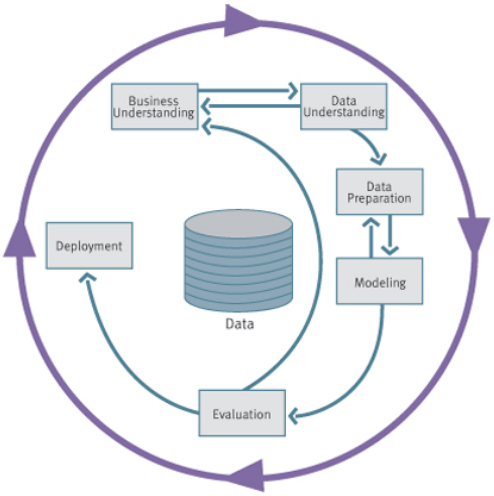


Figure 3.6.: CRoss Industry Standard Process for Data Mining (CRISP-DM). Figure taken from [Wirth, 2000]

Business Understanding This initial phase copes with understanding the objectives and requirements of the problem from a business perspective. As a result the concrete learning task of the data mining problem gets defined and a rough project schedule is developed.

Data Understanding In this phase the given data is collected and analyzed in order to verify the data quality and explore first insights. This might include statistical analysis or subset detection. At the end hypotheses are formulated. These hypothesis then have to be proved or refuted in the further process.

Data Preparation Here the final dataset for learning is constructed. Common tasks are data cleaning, feature extraction and creation of new attributes and the selection of features.

Modeling In this phase, a selection of modeling techniques and learning algorithms are applied on the data. Model parameters are assessed and methods of the evaluation of the gotten models are conceived.

Evaluation This phase is all about evaluating the results: The data mining results are assessed with regard to business success criteria.

Deployment The last step of the process covers the deployment of the models. This includes the presentation of the results in a final report and the presentation for the user. In most cases the user, and not the data analyst, will then be the one to implement the model.

Especially in the third to fifth phase (Data Preparation, Modeling and Evaluation), certain tasks reoccur frequently in practice. Therefore it makes sense to use a tool for simplifying these tasks of the data mining process. I have chosen two different frameworks for this: RapidMiner (see 3.5.2) and the streams framework (see 3.5.3). RapidMiner is ranked first in a poll by *KDnuggets*[3], a data mining newspaper, in 2010. Furthermore Rapid-I, the company that is maintaining and enhancing the RapidMiner, is one of the project partners in the ViSTA-TV project. The streams framework was first developed as a stream data mining plug-in for RapidMiner [Bockermann and Blom, 2012a], but is available as a stand alone tool as well. Both tools are described in further details in the next two sections.

3.5.2. RapidMiner

RapidMiner, formerly YALE (Yet Another Learning Environment)[Mierswa et al., 2006], is an open-source machine learning environment. It was originally developed by Ralf Klinkenberg, Ingo Mierswa and Simon Fischer at the Artificial Intelligence Group at TU Dortmund University. By now it gets maintained and extended by Rapid-I.

RapidMiner offers a construction kit for machine learning tasks. It provides the user with various operators. Each operator receives, processes and dispatches data. By combining different operators, complex data mining processes can be assembled and executed.

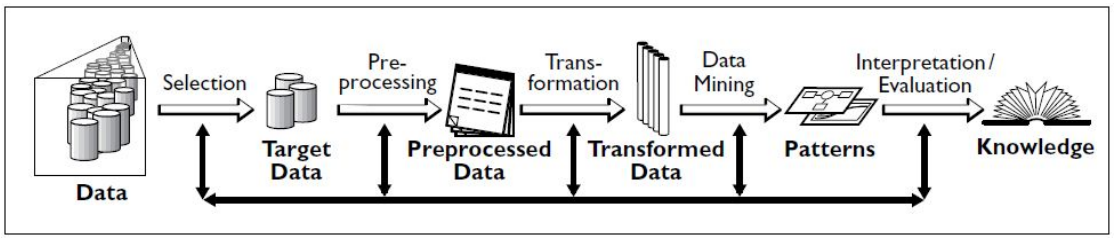


Figure 3.7.: Overview of the steps constituting the KDD process. Figure taken from [Fayyad et al., 1996]

The operators cover all steps of the data mining process (see Figure 3.7). Therefore IO-, Preprocessing- and Transformation- Operators are as well included as Modeling- and Evaluation- Operators. The number of operators can even be increased by installing further PlugIns for certain task or developing your own operators. As we are using

[3] http://www.kdnuggets.com/

RapidMiner to extract image features from video frames, the IMMI-PlugIn for example was useful (see chapter 3.5.2).

RapidMiner is developed in JAVA and used in its current version 5.3.

RapidMiner Image Extension

The Image Mining (IMMI) Extension is an open-source plug-in for RapidMiner (see 3.5.2) extending this data mining platform with image mining functionality [Burget et al., 2010]. It is developed and maintained by a group of researchers from the Signal Processing Laboratory at Brno University of Technology, Czechia,[4] lead by Prof. Zdenek Smekal. In it's current version the extension offers more than 160 operators for processing images with RapidMiner, including IO-, visualization-, preprocessing-, transformation-, feature extraction-, image filter-, and object detection-operators. These functionalities have already been used for various image-related data mining tasks including for example the automatic segmentation of human brain CT images [Uher and Burget, 2012] and the localization of temporomandibular joint disc in MRI images [Burget et al., 2011].

The RapidMiner Image Extension is used for Feature Extraction. A documentation of this can be found in Section 6.2.

3.5.3. Streams framework

The RapidMiner suite is designed to process data in a batch. This means, that static data is processed in one ore more passes over the data. But in many applications, including scientific experiments, sensor networks or live video analysis, the data to be processed is nowadays not static and too large to be stored completely. In this settings, the batch processing model quickly reaches it's limits. I have already pointed out this aspect in section 3.4. The streams framework developed by Christian Bockermann [Bockermann and Blom, 2012b] aims on solving this problem by offering a framework, capable to process data in a streaming manner. In this section I am roughly presenting the framework with particular attention to the processing of video data. For a more general documentation of the streams framework please see [Bockermann and Blom, 2012b].

Streams

A stream is a (possibly infinite) sequence of data items that can be read from various sources. Possible sources are files, databases, or Internet resources, just to mention a few. Implementations for streams that read from .csv-files, SQL databases, JSON or XML formatted data are included in the core streams module. Access to video data is provided by the `stream.io.MJpegImageStream`, a stream implementation that allows to read .raw-files, containing decoded MJpeg images. Figure 3.8 shows the definition for a stream reading video data from an HTTP url.

[4] http://http://splab.cz/

```
<stream id="video"
          class="stream.io.MJpegImageStream"
          url="http://kirmes.cs.uni-dortmund.de/video/20120911-small.raw" />
```

Figure 3.8.: Example for an input stream, consisting of decoded MJepg Images read over the Internet.

Conceptually, a data stream is defined as "an interface, which essentially provides a method to obtain the next item of a stream" [Bockermann and Blom, 2012b]. Each of the data items consist of a set of $(key, value)$-pairs, where *key*s are attribute names and *value*s are the corresponding values. Figure 3.9 shows an example for one data item, containing a serialized image plus some additional information.

Key	Value
`frame`	"stream.image.ImageRGB@2583d260"
`frame:id`	42
`frame:width`	1920
`frame:height`	1080

Figure 3.9.: A data item containing one image plus some additional information.

"The names are required to be of type `String` whereas the values can be of any type that implements Java's `Serializable` interface. The data item is provided by the `stream.Data` interface."

Processors

Processors are "the low-level functional units that actually do the data processing and transform the data items". They basically have one method that receives a data item, manipulates it, and returns the modified (or even a totally new) data item as its result. Furthermore it offers the possibility to perform some initializations when the process gets started. The basic framework for a processor can be found in the appendix.

Processes

The streams framework "defines a *process* as the consumer of such a source of items [=stream]. A process is connected to a stream and will apply a series of *processors* to each item that it reads from its attached data stream."

Stream processes are defined in a XML. In such a XML the stream source as well as the processors are configurated. Figure 3.10 shows an example for a configuration XML.

```
<container>
   <stream
      id="video"
      class="stream.io.MJpegImageStream"
      url="http://mattis.special-operations.de/video/20121027-scaled.raw" />

   <process input="video" >
      <stream.DataItemProcessor1 />
      <stream.DataItemProcessor2 parameter1="42" parameter2="Test" />
   </process>
</container>
```

Figure 3.10.: Example for a configuration XML-file.

The input stream has an unique identifier (id). Each data item of the input stream stream gets processed by a series of processors, where the result of one processor forms the input of the next processor (pipes-and-filters concept).

The whole experiment can be run by calling the `stream.run.main()` function included in the streams project. This function expects a configuration file as a parameter. Afterwards the processors are started, and the `init()` function of each processor is executed. Then the first data item is read and passed to the first processor. The source code to start an experiment from the command line is shown in Figure 3.11.

```
>> java -cp $classpath stream.run experiment.xml
```

Figure 3.11.: Command to run an experiment defined by the corresponding configuration file experiment.xml.

4

Learning tasks

This thesis is about the identification, extraction and evaluation of features from video data. Video data gets produced in countless settings, including television broadcasts, Internet videos, and surveillance cameras. As the results of this thesis are supposed to be used within the ViSTA-TV project, I decided to base the biggest part of my thesis on the analyze of video features, gained from IP-TV data. Nevertheless, the features extraction process and the evaluation of the extracted features are similar for other settings as well. Hence, I decided to build up another use case, that is closer related to surveillance than television. For both use cases, the corresponding learning tasks are defined in this chapter.

4.1. Use case A: IP-TV

As mentioned in the introduction, one important task of the ViSTA-TV project is to understand the interests and behavior of IP-TV users. This shall then be used to predict user switches. To solve this task, it is necessary to have detailed information about the broadcasted television program. Some information about the program already comes with the EPG data, which is provided by the broadcasting companies. We are able to receive the EPG data through our ViSTA-TV project partner *Zattoo*. Unfortunately the EPG data is limited to information on the show level, including

start and **end** time of a broadcast, accurate to the nearest second,

title of the show. In case the show is part of a series, the **episode_title** is also given,

categories the show falls into. This includes for example documentation, news, movie, sports, ...

Hence the segmentation and tagging of the incoming video stream on the show level is already given. But in order to gain better predictions on user behavior and interests, we are interested in a closer grained segmentation. Thus, I decided to take the segmentation

of video data, in particular news shows, into shots and stories as one use case for this thesis. My decision to focus on news videos is due to the fact, that "news videos receive greater attention by the scientific community than all the other possible sources of video" [De Santo et al., 2007]. This interest is mainly due to the fact that broadcasting companies produce a great amount of video material and the reuse of the material is often useful. Hence a good segmentation of news video data is enormously important, as it is the basic step for building up a digital database.The learning tasks are accordingly defined as follows:

- **A1:** Identify all shot boundaries in an incoming news stream. As a measure for success I have chosen the recognition rate, given by precision and recall, as appropriate evaluation criteria. "The recognition rate is the most used quality criterion in order to compare shot change detection methods." [Smoliar and Zhang, 1994].
- **A2:** Recognize all anchorshots. Where does one news story end and the next starts? The success of an inferred classifier can be measured by its' accuracy.
- **A3:** Tag all found segments with meaningful keyword. Even if a keyword is valid for a whole segment, it gets assigned to every single frame included in that segment. Again the accuracy of the assigned keywords can be taken as an evaluation criterion.

I hope that this level of granularity enables us to find explanations for program switches of users and will hence be useful for the further progress of the ViSTA-TV project.

4.2. Use case B: Coffee - Project

Beside IP-TV, surveillance cameras, traffic supervision cameras and web cams do also produce massive amount of video data, that has to be analyzed in real-time. As this thesis is not exclusively limited to IP-TV, I decided to take a second use case into account: Our office is equipped with a *Nespresso* coffee maker. This machine uses differently colored coffee capsules to produce coffee with different flavors. Unfortunately, the consumption of coffee flavors in our office varies a lot, which makes the reordering of coffee capsules difficult. Someone has to count the left over coffee capsules first and estimate the flavors that are missing. The idea of the Coffee project is, to count the consumed capsules automatically by installing a web cam on top of the coffee machine. This web cam will then detect the event of inserting a new capsule, recognize the capsules color and update a counter for each capsule type. Figure 4.1 shows the experiment set-up.

The Coffee project was inspired by the well-known Trojan Room coffee pot camera [1], which was installed at the University of Cambridge between 1991 and 1993. This coffee maker, located in the so called Trojan Room at the Computer Laboratory of the University of Cambridge, was "the inspiration for the world's first web cam". It provided an 128 $\times$128 pixel grayscale picture of the coffee maker within the local network of the university. This enabled everybody working at the Computer Lab to check the status

[1] `http://en.wikipedia.org/wiki/Trojan_Room_coffee_pot`

of the coffee maker from their computer before actually walking to the Trojan Room. Hence useless trips to the coffee maker could be avoided.

Figure 4.1.: The experiment experiment set-up for the coffee capsule project. A web cam, installed on the coffee maker, continuously films the closing lever of the machine. If a capsule is inserted, the event can be detected and the capsule can be classified by its color. The web cam is connected to a computer, processing the incoming video data and making it available over TCP/IP.

The approach to count coffee capsules automatically is -like Trojan Room coffee pot camera- a helpful and demonstrative application to apply techniques, which are useful for other tasks as well. To be able to count the coffee capsules automatically, I have to recognize the capsules that are inserted in the machine. Hence the learning tasks of this project are

- **B1:** Identify all events, where a new coffee capsule is inserted in the coffee maker by supervising the incoming video stream. As events usually last for more then one frame, it is of course not necessary to detect all frames covered by the event. An event is rather said to be detected, as soon as at least one of the frames covered by the event is detected. Again precision and recall are chosen as appropriate evaluation criteria.

- **B2:** Recognize the type of the capsule by identifying its' color. In this setting I am again interested in the percentage of correctly classified capsules based on all capsules. Hence the accuracy is chosen as the suitable evaluation criterion.

The corresponding experiments, which were performed to solve the defined learning tasks, are described in chapter 7.

5

Data

In this chapter I describe how the video data used for the experiments in chapter 7 was gained. The chapter slips up in two parts: In the first section I introduce the ZAPI, that allows us to receive live IP-TV streams over the Internet. The second section is about the creation of the video datasets used for my later experiments.

5.1. Receiving live video data: ZAPI

Zattoo offers their partners a standardized Application Programming Interface (API) called ZAPI (*Zattoo* API). The ZAPI gets used by external developers as well as *Zattoo* itself to develop applications that have to access any kind of content provided by *Zattoo*. This includes a full access to all broadcasted IP-TV channels as well as further program information and EPG-data. The communication with the ZAPI bases on the Hypertext Transfer Protocol Secure (HTTPS). ZAPI receives HTTPS-Requests (POST- and GET-Requests), accepts them and responds the results in either XML (Extensible Markup Language) or JSON (JavaScript Object Notation) format. Thereby it is accessible on almost any device that is capable of doing video streaming.

As I am operating on the actual video stream in this thesis, this chapter gives an overview of how to access the ZAPI and read the video stream. Thus the following chapter structures into how to

- start a ZAPI session
- authenticate with the given user account
- receive the video content of a certain channel
- logout and stop the session

All information are according to the ZAPI documentation that can be found under the URL `http://developer.zattoo.com`. To get access to this developers partner site, you have to contact the Zattoo team and request login information.

5.1.1. Start a ZAPI- Session

A session is a sequence of messages exchanged between an application and the ZAPI. Each ZAPI session gets initialized by a "Hello"-message. This is a POST-Request to the URL `https://HOST/zapi/session/hello`. By this request the client's Application ID (App ID) and a unique ID of your device (UUID) are registered. Furthermore the user has to choose a language and (as a optional parameter) a response format. Possible response formats are XML and JSON. Figure 5.1 shows the source code of a corresponding Java method.

```
private void startSession(HttpClient httpClient) throws Exception {
   HttpPost post = new HttpPost("https://" + HOST + "/zapi/session/hello");

   //Specify Parameters
   List<NameValuePair> params = new ArrayList<NameValuePair>();
   params.add(new BasicNameValuePair("app_tid", app_tid));
   params.add(new BasicNameValuePair("uuid", uuid));
   params.add(new BasicNameValuePair("lang", "de"));
   params.add(new BasicNameValuePair("format", "XML"));

   post.setEntity(new UrlEncodedFormEntity(params));

   //Execute Post
   String response = EntityUtils.toString(httpClient.execute(post).getEntity());
}
```

Figure 5.1.: Java Code for posting a request to ZAPI to start a new session

ZAPI will now respond on the request with a message, containing an XML-response. Figure 5.2 shows an example for a response to a successful request. Afterwards a connection is established and a session is started. In addition a Session ID gets generated. The Session ID is used to identify the session during the following requests.

```
<result>
  <success>True</success>
  <session>
    <active>True_</active>
    <loggedin>False</loggedin>
    <language>en</language>
    <aliased_country_code>CH</aliased_country_code>
    <block_size>3</block_size>
    <min_connectivity>wifi</min_connectivity>
    <current_time>2011-08-23T16:06:06Z</current_time>
    <general_terms>http://zattoo.com/terms-policies</general_terms>
    <privacy_policy>http://zattoo.com/privacy-policy</privacy_policy>
    <recording_eligible>True</recording_eligible>
    <upgrade_available>True</upgrade_available>
  </session>
  <urls>
    <url>http://Zattoo.ivwbox.de/cgi-bin/ivw/CP/DE-ANONYMOUS</url>
  </urls>
</result>
```

Figure 5.2.: Response to Start-Session

5.1.2. Login with user account

Although users may be able to access certain Zattoo content without having a Zattoo account or being logged in, most services require a registration. Registered users get a login and a password and certain permissions are stored for each account. To log in a user another POST-Request request has to be sent to the URL `https:////HOST/zapi/account/login`. This request includes the user name (login) and the password as parameters. ZAPI will again answer with a message, containing an XML-response. Figure 5.3 shows, how this response will look like.

```
<result>
  <success>true</success>
  <account>
    <login>username</login>
    <services/>
    <products/>
    <subscriptions/>
    <max_playlist_size>250</max_playlist_size>
    <pvr_trial_expiration>2012-11-14T19:34:00Z</pvr_trial_expiration>
  </account>
</result>
```

Figure 5.3.: Response to User Login

As you can see, all services, subscriptions, and products, the user is subscribed to, will be listed in this response. After receiving the successful response message, the user will now be able to access all content that is available for him over the established session.

5.1.3. Watch a channel

To receive the actual stream of a channel's video and audio data, a further POST-Request has to be sent. This request must be addressed to the URL `https://HOST/zapi/watch` and contains the requested stream type and channel ID (cid) as parameters. Possible stream types are HTTP Live Streams (HLS) or Silverlight Smoothstreams, where HLS8 is way more common. The channel ID is a unique text identifier of a channel. Figure 5.4 again shows an example for a successful ZAPI response.

```
<result>
  <urls> ... </urls>
  <stream>
    <url>http://HOST/ZDF-live.m3u8?maxrate=0&amp;uuid= ... </url>
  </stream>
  <success>True</success>
</result>
```

Figure 5.4.: Response to Watch Channel

This response contains the URL, under which a playlist file can be found. Playlist file can be received by sending a GET-Request to the given URL. These playlist files are

plain text files in the .m3u(8)[1]-format, containing the locations of the actual media files. Following the URL given in the watch channel response, we receive a .m3u-file, that again refers to other .m3u-files. An example for such a response can be found in figure 5.5.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=64000
ZDF-live-64.m3u8?watchid= ...
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=150000
ZDF-live-150.m3u8?watchid= ...
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=300000
ZDF-live-300.m3u8?watchid= ...
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=600000
ZDF-live-600.m3u8?watchid= ...
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1100000
ZDF-live-1100.m3u8?watchid= ...
```

Figure 5.5.: The .m3u-file received, when following the link given in the watch channel response. This file routes the user to the correct playlist file, optimized for his bandwidth.

Each given .m3u-entry in this file is again a playlist file, which is optimized for different bandwidths. Hence users with faster internet connections get directed to a playlist containing media elements with higher quality. After choosing the playlist, which is appropriate for the available bandwidth, the right .m3u-file can be downloaded using another GET-Request. This received playlist file now is a media playlist according to the HTTP Live Streaming (HLS) Protocol [Pantos and May, 2009]. An example can be found in figure 5.6.

```
#EXTM3U
#EXT-X-TARGETDURATION:4
#EXT-X-MEDIA-SEQUENCE:67978019
#EXT-X-KEY:METHOD=AES-128,URI="http://HOST/ZDF/live/64/6421.key?watc ..."
#EXTINF:4,
http://HOST/ZDF/live/64/67978019s.ts?__gda__= ...
#EXTINF:4,
http://HOST/ZDF/live/64/67978020s.ts?__gda__= ...
#EXTINF:4,
http://HOST/ZDF/live/64/67978021s.ts?__gda__= ...
#EXTINF:4,
http://HOST/ZDF/live/64/67978022s.ts?__gda__= ...
#EXTINF:4,
http://HOST/ZDF/live/64/67978023s.ts?__gda__= ...
```

Figure 5.6.: The .m3u-file received, when following one of the links given in the first .m3u-file. This file contains references to the media segments.

This .m3u-playlist refers to a set of media segments, encoded in the MPEG-2 Transport Stream (TS)-format. All media segments can be downloaded from the *Zattoo* server and each of the segments contains four seconds of video and audio data. When putted

[1].m3u-files are usually encoded with the Latin-1 charset. The file ending 8 indicates that this file is the unicode version of .m3u, which uses UTF-8 unicode characters

together in the right order (given by the ascending numbering of the segments), we obtain a full video and audio data stream of one channel. The .m3u-file gets updated regularly, whereby new elements get added and old elements get deleted. Hence the video stream is infinite. In order to ensure that only registered users watch the television channels, the .ts-files are AES-encrypted and have to be decrypted before putted together.

5.1.4. Stop Channel, logout and stop session

After all the desired data is received, the channel should be stopped, the user should be logged out and the session should of course be terminated. This is done by three easy POST-Requests; each of them not requiring any parameters. The request have to be addressed to the URLs `https://HOST/zapi/stop`, to stop the channel, `https://HOST/zapi/account/logout`, to log out the user and `https://HOST/zapi/session/goodbye`, to stop the session.

5.1.5. Error responses

In some cases the sent requests may be irregular or can not be processed by the server. In these cases the server replies with an error response, providing the user with brief information about the occurred problem. An example for such an error-response can be found in figure 5.7.

```
<result>
  <success>False</success>
  <http_status>400</http_status>
  <internal_code>2</internal_code>
</result>
```

Figure 5.7.: Example for an error-response

The error code can be looked up on the Zattoo website, providing the user with hints how to fix the error.

5.2. Creation of datasets

In order to be able to evaluate the performance of the applied approaches for shot boundary detection, anchorshot detection, and coffee capsule recognition, it is necessary to have labeled data. Hence it is not possible to test the approaches on live video data received over the ZAPI. Rather we need to have fixed video datasets: one containing the video data of news shows and one for the coffee capsule recognition project.

For this purpose, we downloaded video files of news shows on the Internet respectively recorded a video file containing data of different coffee capsules. Unfortunately neither the RapidMiner Image Mining Extension (see section 3.5.2) nor the streams framework (see section 3.5.3) offers the opportunity to read in MPEG video files directly: The RapidMiner Image Mining Extension requires a folder containing JPEG images as its input, whilst the MJpegImageStream implementation of the streams framework (see

section 3.5.3) requires a file, containing RAW decoded video frames in the JPEG image format. Hence the MPEG video files need to be converted into the desired formats. There is plenty of good software for uncompressing media files available. I only want to mention the two tools I actually used: xuggler and mencoder.

xuggler is an open-source tool to uncompress, modify, and re-compress media files from Java. It is available on the xuggler project homepage[2] and distributed under the GPL Version 3 license. It consists of a Java part and a set of native libraries and uses FFmpeg to compress and uncompress media data. The usage is quite simple: The java class DecodeAndPlayVideo.java has a static main method. When calling this main method with a string parameter, which represents a filename, the video file is opened, gets uncompressed and displayed. As all that functionality is written in Java, this can easily be modified and the uncompressed JPEG image can be stored on the disk. Figure 5.8 shows the invocation command for the DecodeAndPlayVideo class.

```
File videofile = new File ("C:/tagesschau.mp4");
DecodeAndPlayVideo.main(videofile);
```

Figure 5.8.: Invocation command for uncompressing a video file using xuggler

mencoder[3] is a free command line tool for video decoding and encoding. It is related to the MPlayer (which is similar to FFmpeg) and included in all MPlayer distributions, but provides a standalone version as well. Released under the GNU General Public License, it is available without charge for various operating systems, including Linux and MacOS. The mencoder can convert a great amout of media data formats into each other, including MPEG and RAW JPEG sequence files. Hence it enables us to convert the downloaded video data into a format, which can be read in by the streams framework with the MJpegImageStream. The command needed to invoke mencoder is shown in figure 5.9. The scale command is optional and allows to redefine the size of the frames of the resulting output data.

```
>> mencoder videofile.mp4 -nosound -of rawvideo -o videofile.raw
         -ovc lavc -lavcopts vcodec=mjpeg [-vf scale=640:360]
```

Figure 5.9.: Invocation command for uncompressing a video file using mencoder

5.2.1. The "news"-dataset

For the evaluation of the learning tasks A1, A2, and A3, I use a dataset consisting of four "Tagesschau" news show downloaded from *YouTube*. The "Tagesschau" is the best-known German news show, broadcasted on the first German program (ARD) every

[2] http://www.xuggle.com/xuggler
[3] http://mplayerhq.hu/

night from 8:00pm to 8:15pm. The reason for downloading the videos from *YouTube* is, that they have the identical video format as the onces we receive when using the ZAPI. The advantage is, that I did not have to glue together the MPEG Transport Stream (.ts) segments, which made the process of creating the dataset a little bit easier. The downloaded videos are listed in tabular 5.1.

Date	**Length** in minutes
Tuesday, September 11th 2012	15:38
Thursday, September 13th 2012	15:44
Saturday, September 15th 2012	12:21
Saturday, October 27th 2012	13:36
Total	57:19

Table 5.1.: Video data included in the "news" dataset

Each of the video files has a resolution of 1280 × 720 pixels per frame (HD ready). In order to be able to process the data in RapidMiner,the video files were first encoded using xuggler. As the result I get a folder containing images of each single frame (see figure 5.10. Based on the encoded data, I attached labels to the resulting JPEG images. The first multinominal label is "cut" (C), "gradual transition" (GT), and "not cut" respectively. The second label holds information about the type of the shot. It distinguishes between "anchorshots" (AS) and "no anchorshots". The "no anchorshot" class is further labeled as "intro", "extro", "weather" and "n_a", which covers all sort of news report shots including interviews and commentaries. By using mencoder,the video files have furthermore been converted into a format that can be imported in the streams framework. The video files as well as the .csv files containing the labels are available online. The url is `http://mattis.special-operations.de/video/`. The labels of one news show are furthermore included in the appendix.

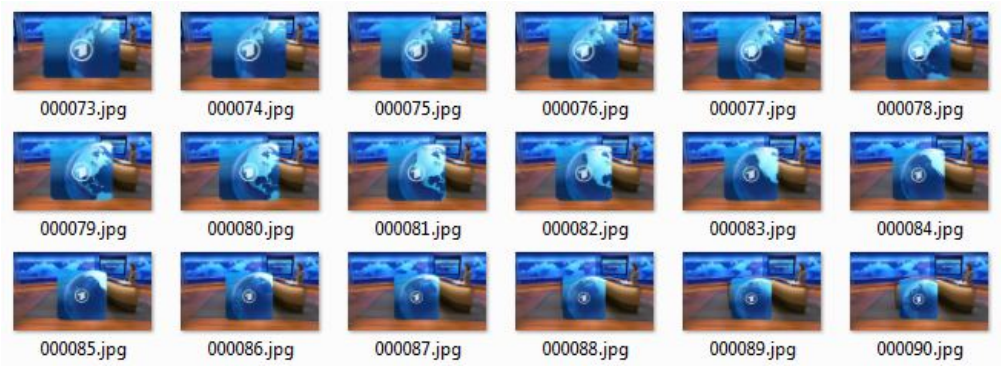


Figure 5.10.: Images resulting from decoding a video with xuggler. Each frame gets stored as a JPEG image.

5.2.2. The "coffee"-dataset

For the evaluation of the learning tasks B1 and B2, I use a dataset consisting of two short video sequences. Each of the video files shows 36 coffee capsules, which slip down a slide. A capsule slipping down the slide is hence one "event", whilst the empty slide can be seen as "no event". The video files are recorded at a stretch, using a webcam with a resolution of 640 × 480 pixels per frame. 25 frames per second are captured. The 36 coffee capsules split into six different classes, representing six different coffee flavors. All flavors can be identified by the color of the capsules. Possible colors are "red", "yellow", "blue", "green", "black", and "purple".

I have decided to produce the labeled dataset by slipping coffee capsules down a slide for two reasons: First of all, the *Nespresso* coffee maker in our office broaches all coffee capsules that are inserted. Hence it would not have been possible to create an artificial dataset within a short amount of time, without having to drink a whole lot of coffee. Furthermore I was not sure, if I would be able to manually determine the color of coffee capsules, which where inserted over time, in order to label the data correctly. Especially purple, dark green, and black capsules did almost look the same to me. So I decided to create the dataset artificially. The inferred techniques for event detection and capsule recognition will afterwards be transferable to the real setting easily.

Beside the video data, the dataset contains files, which hold the true labels for both of the video files. This includes the number of the frame, where an event occurs, plus the color of the coffee capsule, which is slipping down the slide at that moment. As an event spans more than one frame, only the first frame of the event is stored in the label file. Implicitly the following six frames belong to the same event. Figure 5.11 shows an example for one capsule event.

An example for a file, containing the manually assigned labels for one of the two video sequences, is included in the appendix. The full dataset, including the video data as well as the label files, is available online. The url is `http://mattis.special-operations.de/video/`.

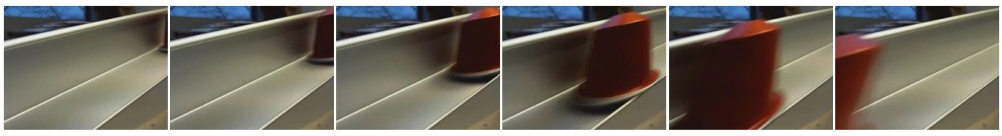


Figure 5.11.: A red coffee capsule slipping down the slide. All six consecutive frames belong to the same capsule event.

6

Feature Extraction

Videos are basically a sequence of images (*frames*) and short audio-samples. Therefore, most methods developed for learning on images can also be applied to video data. The following section gives an overview of the features I have extracted from the video data for this diploma thesis. Since the extraction of features from images is a common task in computer vision and countless approaches have been developed over the years, I can only focus on a small subset of all possibly extractable features. Of course, the following collection is surely not exhaustive and there might be useful features I have not identified yet, but it will turn out, that the features extracted so far are sufficient to solve the specified learning tasks. Nevertheless the extraction of further features might turn out to be necessary during the next phases of the ViSTA-TV Project.

In order to evaluate which image features are useful for solving the defined learning tasks (see chapter 4), I have first of all extracted a huge amount of features using RapidMiner. The RapidMiner Image Mining Extension takes JPEG images, gained from the video data by using xuggler, as its' input and enables us to extract and evaluate features quickly. The features extraction is described in section 6.2. As RapidMiner does not provide the user with any stream functionality, I have afterwards developed processors for the streams framework, enabling me to extract those features, which turned out to be useful, on live video data in real-time as well. An overview of the implemented methods can be found in section 6.3).

6.1. Naming convention for extracted features

In order to keep track of the huge amount of features that might potentially be created using all sorts of available data (i.g video, audio, EPG and user-data), a consistent naming convention is necessary. Therefore we have developed a hierarchical naming scheme. We have decided to separate each level of the hierarchy by using a ":". On the first level of the hierarchy we should tell, from which kind of data we gained the feature. Possible values are: "video:", "audio:", "epg:", "user:" or "dbpedia:", just to mention a

few examples. The second level of the hierarchy then of course depends on the first level. For "video:" we have decided to specify next, whether the feature is related to a single frame ("frame:"), a sliding windows of n frames ("n-frames:") or a whole show ("show:"). As difference images of two successive frames are useful for cut detection, the "2-frame:"-features are especially interesting. Furthermore we hope to be able to calculate them very quick by taking advantage of the MPEG-codex (see chapter 2.2.1). For "video:frame"-features the next level of the hierarchy than describes the color channel the feature is gained from. Image 6.1 shows an extract from the full naming space for features used in my diploma thesis. This proposal has been adopted for the ViSTA-TV project.

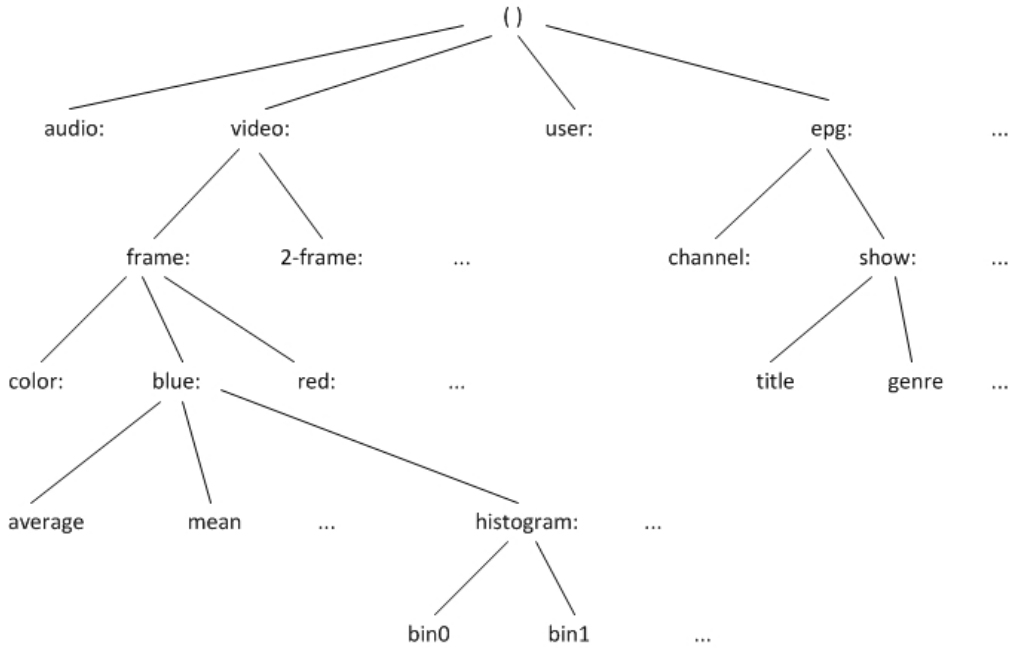


Figure 6.1.: The naming space for features.

6.2. Image Feature Extraction with RapidMiner

The RapidMiner Image Mining Extension (see 3.5.2) provides a bunch of useful operators for extracting features from single images. These include operators for image transformation, filtering and feature extraction. Furthermore it comes with an operator, allowing us to loop over all images contained in one folder (*Multiple Color Image Opener*). As we are able to decode a movie file and store all images in one folder by using xuggler (see 5.2), we can not only use the Image Mining Extension to extract features from images, but from video data as well. In the following section 6.2.1, I first of all describe the IMMI operators used for the feature extraction. Afterwards the whole feature extraction process in RapidMiner is explained.

6.2.1. IMMI Operators

Image Representations

Digital images can be considered as a two-dimensional matrix, which contains a fixed number of rows and columns. The values in this matrix represent the pixels of the image. Based on the type of image, the values in the matrix are usually either one bit, one byte, or eight byte long, allowing us to represent different color types.

Monochrome Images In monochrome images (sometimes also referred to as grayscale images) these pixels are represented by just a single one byte value per pixel indicating the brightness. Usually we have one-byte values, allowing us to distinguish between values ranging from 0 to 255.

Color Images Whereas monochrome images can be represented by just a single byte per pixel, color images require at least three values. The most common color space in this context is the RGB color space. Here a colored image is represented by three values representing the relative proportion of the three primary colors of light: red, green and blue. Figure 6.2 gives an example for the decomposition of a color image into its three color channels. In the Figures (b) to (d) a dark colored pixel indicates a high value on the same pixel on the corresponding color channel. For example as the sky is blue, the sky becomes darker in the blue channel image then on the red and green channel images.

Other color spaces such as YCbCr are not taken into account in this thesis.

Black and White Images Black and White images (sometimes also referred to as binary or bi-level images) are a special subtype of monochrome images that only have two gray levels: 0 (=white) and 1 (=black). Since there are only two possible values for each pixel, the image matrix to store the image can be a binary one. This saves disks space, speeds up the transmission of the image and makes some evaluations faster.

(a) Original image (b) Red channel (c) Green channel (d) Blue channel

Figure 6.2.: The RGB channels of a color image from the Texas State Capitol in Austin.

Images of one color type are downwardly compatible. That means that color can be transformed into imagemonochrome images, and monochrome images can be transformed into black and white images.

Converting color images into monochrome images

In some cases it might be necessary to convert a color image into a monochrome one. Unfortunately, this conversion is not done uniformly troughout different tools. In all cases the gray value of a pixel is calculated as a weighted sum of the RGB-values of the pixel.

$$GRAY = W_R * R + W_G * G + W_B * B$$

Typical weighings W_R, W_G, W_B are:

- The equalized weighting of all channels is the simplest form. The weights are:

$$W_R = W_G = W_B = \frac{1}{3}$$

- "Since we perceive both red and green as being substantially brighter then blue, the resulting image will appear to be too dark in the red and green areas and too bright in the blue ones." Therefore unequal weights, originally developed for encoding analog color television signals, are more common: [Burger and Burge, 2007]

$$W_R = 0.299, W_G = 0.587, W_B = 0.114$$

- ITU-BT.709 [itu, 2002] even recommends slightly other weights for digital color encoding:

$$W_R = 0.2125, W_G = 0.7154, W_B = 0.072$$

The equalized weighting as well as the second methode are also available in the IMMI *Color to grayscale* operator.

Converting monochrome images into black and white images

Monochrome images can be transformed into black and white images by applying a threshold on the value of each pixel. If the gray value of a pixel exceeds the given threshold, the pixel is set to black or white wise versa. Figure 6.3 shows an example for a color image converted into a black and white image. The intermediate step of converting it into a monochrome image is not shown. The conversion can be performed using the IMMI *Thresholding* operator.

(a) Original image (b) Transformed Image

Figure 6.3.: Image transformed to black and white with threshold.

Image Transformations

Filtering

In certain settings, the appliance of image filters might be a helpful technique to improve the quality of the extracted features. For instance if the input image contains a lot of noise, it can be useful to do some image smoothing first. Therefore this section gives an overview of some popular image filtering methods.

Image Smoothing Each pixel in a frame is replaced by a new value, based on its neighborhood. In the simplest case, the new value is calculated as the unweighted average of the color values of all eight nearest neighbors plus the pixel itself (3×3 mask). This is shown in figure 6.4 (a) [Bhabatosh Chanda, 2011]. More complex approaches use bigger masks, even not necessarily shaped quadratically, weighted averages or the other statistical approximations like medians, maximum, or minimum values. Figure 6.4 (b) shows an example for a 5×5 *Gaussian*-weighted mask.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(a) Unweighted 3×3 mask

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

(b) Gaussian-weighted 5×5 mask

Figure 6.4.: Examples for two different image smoothing masks. In a Gaussian-weighted masks the weights of each pixel (x, y) are calculated by the two-dimensional Gaussian function $f(x, y) = exp(-(x^2 + y^2)/s)$ where s is the scale parameter controlling the spread [Bhabatosh Chanda, 2011].

The problem of simple averaging filters is, that they tend to blur edges and details. This can be overcome be either using median filters or order-statistics filters

[Rafael C Gonzalez and Woods, 2008], or applying other filters, that sharpen images.

Image Sharpening Whenever edges, line structures or other details in an image are important, image sharpening filters are used to enhance these structures. These filters try to detect edges and increase the intensity of the detected edges in the resulting image. For further information about edge detection please see chapter 6.2.1. The big challenge is to enhance edges without increasing noise at the same time. Although there are existing approaches (i.e. [Alvarez et al., 1992]), the problem is still an open field for further research.

IMMI offers miscellaneous filters. All of them are available over the *Image Filters* package.

Border detection

The detection of borders (also called "edge detection") in an image might be a useful method in order to get abstract information about the image. The border detection algorithm implemented in IMMI operates on color images. These color images are first of all transformed into a color-space containing 64 colors. This is done by uniformly quantizing each of the RGB color channels into four quantiles. Afterwards each pixel, having all four neighbors (top, bottom, left, and right) the same quantized color, is classified as an interior pixel. All other pixels are classified as border pixels. The outcome is a black and white image, where all border pixels are black and all interior pixels are white. Figure 6.5 shows an example for this. The correspondent operator is called *Border / Interior Classification* (short: BIC) operator.

(a) Original image (b) Transformed Image

Figure 6.5.: Example for border detection on a color image.

Difference images

As RapidMiner offers no functionality to operate on video data, there is no way to benefit from any features that can be gained from the MPEG compression. Unfortunately some of the presented approaches for shot boundary detection base on these features, especially the difference image of two frames. Therefore I "emulated" the MPEG compression in RapidMiner by reproducing the difference image of two frames and extracting features on these difference images. The corresponding process is shown in figure 6.6.

First of all both frames are loaded as monochrome images using the *Open Gray-scale Image* operator. Then the grayscale values of all pixels in both images have to be multiplied with 0.5. Afterwards 128 is added to all pixel values in one of the images, and last the two images get subtracted using the *Image Combinator* operator. An example for the output of this process can be found in figure 2.5.

Global image statistics

So far I have only described operators that transform images. The next two paragraphs now cope with the actual feature extraction.

The IMMI Extension offers an operator called *Global Feature Extractor from a Single Image* to extract global statistics or histograms from images. Table 6.1 shows an overview of all features that can be extracted by the *Global Statistics*-Operator.
The output of the operator is an ExampleSet containing values for all requested features. I have applied this operator on the original image, the grayscale image, each color channel, the reverse engineered difference image, the black and white image, and an image resulting from the border detection separately. Further details are described in the following section 6.2.2.

Histograms

Beside the *Global Statistics*-Operator, another interesting operator for extracting information from an image is the *Histogram*-Operator. A color histogram represents the distribution of colors in an image. After defining the number of bins, color ranges, which cover the whole color space of the image, are derived automatically. Then each pixel is sorted into the corresponding bin and afterwards the number of pixels in each bin is counted. The relative count of pixels belonging to a bin then is the value for this bin. The color histogram therefore can be seen as an statistical approximation of the continuous distribution of color values for a frame.

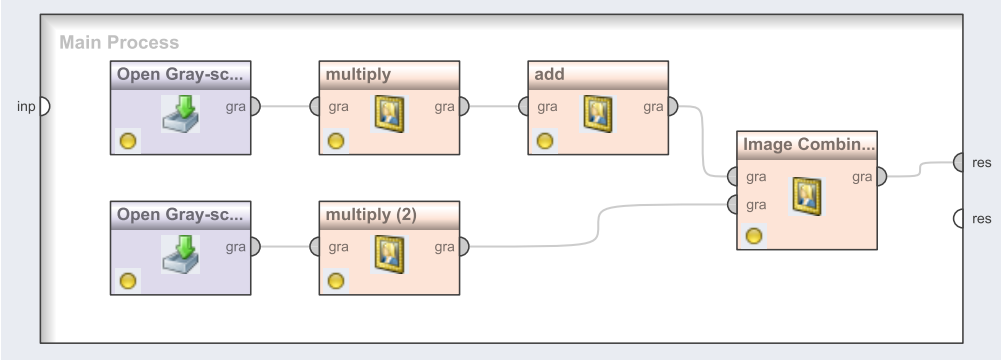


Figure 6.6.: The RapidMiner process for reproducing the difference image of two frames.

Mean	The average gray value of the input image.
Median	The median gray value of the input image.
Standard Deviation	The standard deviation of the gray values.
Skewness	The skewness of the gray values.
Kurtosis	The kurtosis of the gray values.
Peak	Include peak color from histogram and its relative value.
Min Gray Value	Include Minimum Gray value in the image.
Max Gray Value	Include Maximum Gray value in the image.
Center of Mass	Delivers two features X and Y, denoting the X- and Y-coordinate of the brightness-weighted average of the X and Y coordinates of all pixels in the image.
Normalized Center of Mass	Delivers two features X and Y. Same as Center of Mass, but normalized, so that the size of the image doesn't matter.
Area Fraction	The percentage of non-zero pixels.
Edginess	Relative count of white (255) pixels after applying "Find edges" operator.

Table 6.1.: Overview of image features that can be extracted using the *Global Statistics*-Operator of the IMMI Extension

(a) Original image (b) Color histogram

Figure 6.7.: Example of a color histogram for a "Tagesschau" frame.

6.2.2. The overall feature extraction process in RapidMiner

Using the above described operators, I designed a RapidMiner experiment to extract video features from a set of images, extracted from a video file by using xuggler. A screenshot of the main part of the experiment is shown in figure 6.9.

By using the *Multiple Color Image Opener*, I loop over all images in the folder created by xuggler. The process shown in figure 6.9 is the inner process of the MCIO-operator and is hence executed for every image. First the image gets multiplied seven times. One of the seven images is kept as it is, the others are transformed by selecting a single color channel only, calculating the difference image, converting the image to grayscale, or applying border detection. Afterwards the *Global Feature Extractor from a Single Image*-operator is applied to each of the resulting images in order to extract features from the images. In order to save computational time, the operator is not always used to its' full extent. For example color histograms are only extracted of the original image. After renaming the extracted features and adding an id in order to be able to join the resulting example sets, the extracted features are written into a .csv-file. As the result of the whole process, I gained a set of 80 features per frame. A complete list of the features extracted with the RapidMiner IMMI is included in the appendix.

Figure 6.8.: Multiple Color Image Opener

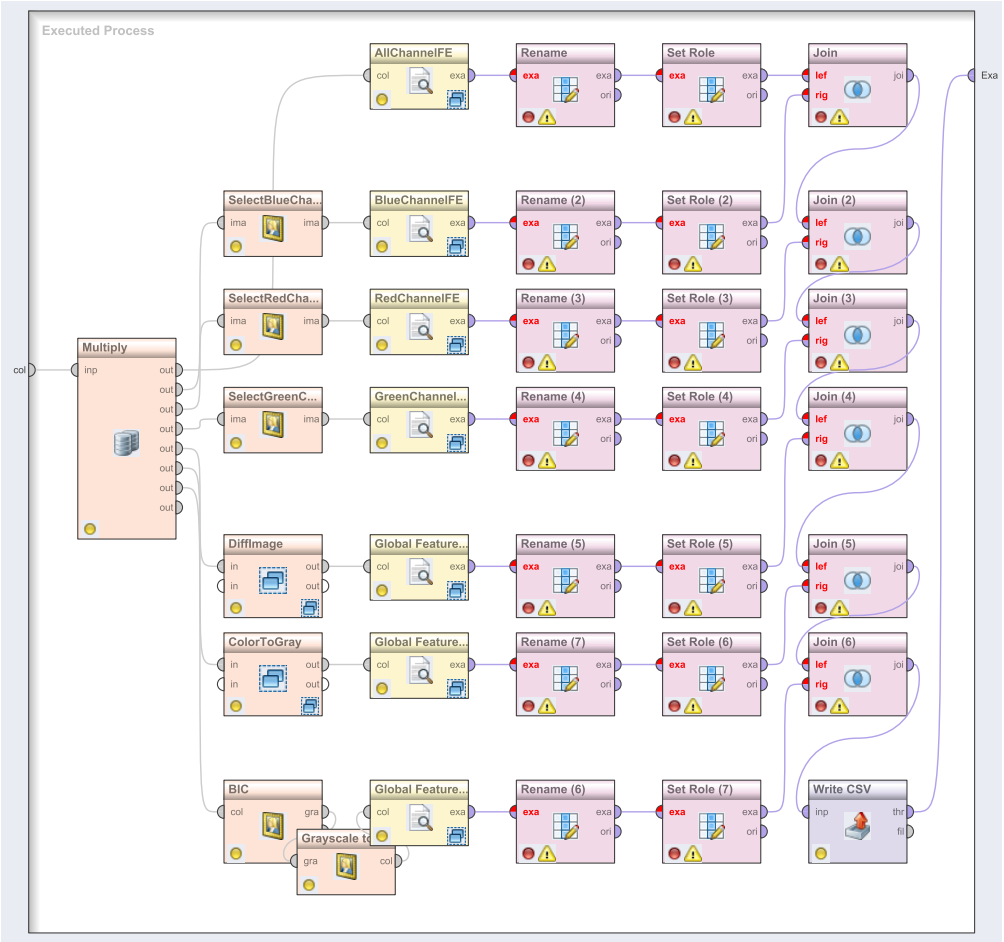


Figure 6.9.: The overall feature extraction process in RapidMiner

6.3. Image Feature Extraction with the streams framework

RapidMiner does not provide the user with any stream functionality. But given that video data should be processed in real-time, we need to extract features in a streaming manner. This can be achieved by using the streams framework (see section 3.5.3). The following section gives an overview of the processors, which I have implemented for processing video data and extracting features. A full list of all implemented processors, including more details about parameters and the API, can be found in the appendix.

6.3.1. Representing images

By using `stream.io.MJpegImageStream`, I can read in decoded video data and create a stream of data items. Each of these data items corresponds to one frame of the video stream. It contains the frame itself as a serialized image plus some additional information. As the image type `java.awt.image.BufferedImage`, which is usually used in JAVA to store images, is not serializable, it can of course not be used. Hence the images included in the data items are of the self-implemented type `stream.image.ImageRGB`. An `ImageRGB` image can be obtained from a `BufferedImage` by using the `ImageRGB (BufferedImage img)` constructor. On the other hand an `ImageRGB` image can be converted to a `BufferedImage` using the provided `BufferedImage createImage()` function.

In RapidMiner image types vary depending on the image representation. Hence IMMI operator can either process color images, monochrome images or black and white images. In contrast to this concept, I decided to represent all types of images by the `ImageRGB` class. Monochrome images can be stored by calculating the grayscale value for each pixel and setting the value of all three RGB channels of the image to this grayscale value. Black and white images get stored by setting each pixel either to black or white. The advantage of this approach is that every image processor can handle every image and the processors can be combined in any order. When using RapidMiner the combination of operators is sometimes difficult, due to the fact that images of different types have to be converted into each other to ensure the outcome of one operator matches the income of the next one.

6.3.2. The AbstractImageProcessor

Data items, containing `ImageRGB` images, can be processed by special processors, extending the `stream.image.AbstractImageProcessor` processor. The advantage of using the `AbstractImageProcessor` instead of the `AbstractProcessor` is that the `AbstractImageProcessor` allows an easy access to the image, included in the data item, without searching for the image first. Nevertheless it is still possible to access the full data item. This is necessary in order to store new features or access further information, that is included in the data item. Figure 6.10 shows the source code of the `AbstractImageProcessor`.

```
package stream.image;

public abstract class AbstractImageProcessor extends AbstractProcessor {

      protected String imageKey = "data";

      @Parameter(description = "The name of the attribute that contains the byte array
          data of the image.", required = true)
      public void setImage(String data) {
            this.imageKey = data;
      }

      public String getImage() {
            return imageKey;
      }

      public Data process(Data input) {

            Serializable value = input.get(imageKey);
            if (value instanceof ImageRGB) {
                  Data result = process(input, (ImageRGB) value);
                  return result;
            }

            byte[] bytes = (byte[]) input.get(imageKey);
            if (bytes == null) {
                  return input;
            }

            try {
                  BufferedImage bufferedImage = ImageIO
                              .read(new ByteArrayInputStream(bytes));
                  if (bufferedImage == null) {
                        log.debug("No valid JPEG image!");
                        return null;
                  }
                  ImageRGB img = new ImageRGB(bufferedImage);
                  Data result = process(input, img);
                  return result;
            } catch (Exception e) {
                  log.error("Error processing image: {}", e.getMessage());
                  if (log.isDebugEnabled())
                        e.printStackTrace();
                  return input;
            }
      }

      public abstract Data process(Data item, ImageRGB img);
}
```

Figure 6.10.: The `AbstractImageProcessor`.

6.3.3. Implemented image processors

Using the `AbstractImageProcessor`, I have implement a wide variety of processors, which can process images. This section just gives an overview of all processors. For a full list please see the appendix. All presented processors are at least able to process images in real-time, given that we are receiving not more than 25 frames, each in Full HD resolution (1920×1080 pixels), per second. This throughput was achieved using my

personal computer, which is specified in more detail in the following chapter 7.

Image Transformation processors

When operating with the RapidMiner IMMI, it has turned out to be necessary to transform images. Hence I have developed a set of image transformation processors.

The **BorderDetection** processor performs a border detection on the input image. A pixel is said to be a border pixel, if its' color differs from the color of at least one of its' four neighboring pixels significantly. By increasing the tolerance, the amount of neighboring pixels that have to have the same color value can be decreased.

The **ColorDiscretization** processor discretizes the color space of the input image by discretizing each single RGB color channel.

The **ColorToGrayscale** processor converts a color image to a grayscale image.

The **Crop** processor crops an image to a new size by cutting away parts of the image.

The **DiffImage** processor computes the difference image of the actual image and the image before.

The **Smoothing** processor is a filter, smoothing the image. Hence the processor reduces the noise in an input image. Each pixel gets replaced by the average of pixels in a square window surrounding the pixel.

Image Feature Extraction processors

The **AverageRGB** processor extracts RGB colors from a given image and computes the average RGB values over all pixels of that image.

The **CenterOfMass** processor calculates the Center of Mass of one color channel of the image. You can either ask for the absolute x- and y-coordinates of the Center of Mass or the normalized Center of Mass (= absolute Center of Mass / the size of the image).

The **ColorHistogram** processor computes a color histogram on one color channel of an image.

The **MedianRGB** processor extracts RGB colors from a given image and computes the median value for each color channel.

The **StandardDeviationRGB** processor computes the standard deviation for all three RGB channels. Requires the Average (=Mean) value for all RGB channels to be included already. This can for example be done by using the AverageRGB processor.

Further processors

Beside the above mentioned processors, I have implement further processors, which were necessary to run the experiments. A list of all processors is included in the appendix.

7

Experiments and Evaluation

Using the approaches presented in chapter 2 and the data described in chapter 5, I now address the tasks of shot boundary detection, news video segmentation, and coffee capsule application as introduced in chapter 4.

All experiments, which are described in this chapter, have been run on my personal computer, equipped with a dual-core Intel Core(TM) i5-2520M CPU with 2.5 GHz clock speed and 4 GB main memory.

7.1. Shot Boundary Detection

As described in chapter 2.2 the detection of shot boundaries is the first important step towards a meaningful segmentation of video content. This detection splits up into two learning tasks: The detection of hard cuts and the detection of gradual transitions. As we have seen in table 2.1, hard cuts seem to be more likely to occur than gradual transitions. Unfortunately this statistics was taken from a comparison paper written in 1996 and talking about TV broadcasts from the United States [Boreczky and Rowe, 1996]. So we should first have a look on current video material to see, whether those results are

Date	**# of frames**	**Cuts**	**Gradual Transitions**
11.09.2012	23.457	130	10
13.09.2012	23.619	130	3
15.09.2012	18.522	100	8
27.10.2012	20.406	110	5
Total	86.004	470	26

Table 7.1.: Frequency of hard cuts versus gradual transitions in todays' "Tagesschau" news shows.

valid for todays' video content of German broadcasting companies as well. Therefore I manually analyzed the four news show videos (approximately 1 hour of video data) of "Tagesschau", included in my "news" dataset. The results are shown in table 7.1.

Obviously todays' news shows still have way more hard cuts than gradual transitions in them. It looks like the rate has even been rising over time. Hence we simplifying focus on the detection of hard cuts first.

7.1.1. Approach and Evaluation

As described in chapter 2.2, the simplest approaches for hard cut detection base on the calculation of pair-wise differences between successive frames. This can be done in the Stream Framework as well, by calculating the difference image, converting the difference image to grayscale and checking, whether the average gray value of the difference image exceeds a threshold T. This approach almost is a direct reimplementation of the approach by [Nagasaka and Tanaka, 1992]. The difference is that T in my implementation is a threshold on the average gray value of the difference image. Thus it does not need to be adjusted to the resolution of the input video stream.

```
<container>

   <stream id="video"
      class="stream.io.MJpegImageStream"
      url="http://kirmes.cs.uni-dortmund.de/video/20120911-small.raw" />

   <process input="video" >
      <CreateID key="frame:id" />
      <stream.news.helper.AddNewsshowLabels />

      <stream.laser.DiffImage threshold="0"/>
      <stream.image.ColorToGrayscale />
      <stream.image.AverageRGB />
      <stream.news.learner.sbdetection.GrayThreshold t="35" />

      <stream.learner.evaluation.PredictionError label="@label:shotboundary" />
      <stream.statistics.Sum keys="@error:shotboundary" />

   </process>
</container>
```

Figure 7.1.: Streams framework process to detect shot boundaries. A shot boundary is declared, when two successive frames differ more than a given threshold. The difference of two successive frames is calculated on the grayscale difference image of the frames.

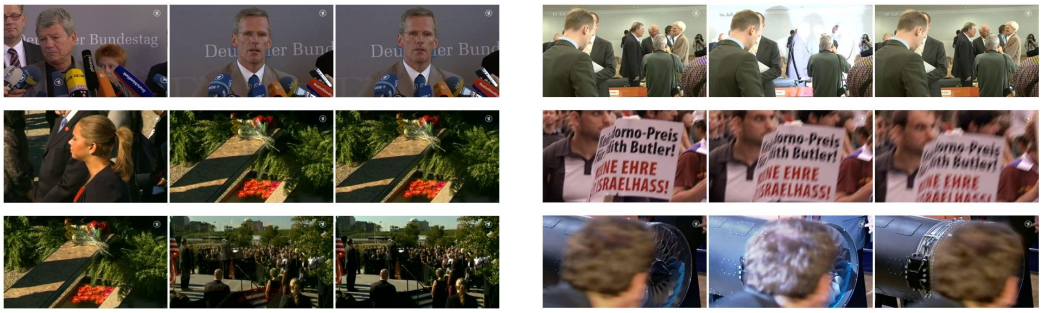
Figure 7.1 shows the stream framework process for the described experiment. The threshold $t = 35$ is chosen randomly. The resulting confusion matrix shows, that 97,7% (128 out of 131) of all cuts have been correctly classified as a cut (True Positives). Unfortunately the number of False Positives (shots that were incorrectly labeled as cuts) is quite high as well (precision = 71.5 %, recall = 97,7%).

	Prediction: true	Prediction: false	Total
Label: True	128	3	131
Label: False	51	23.276	23.327
Total	179	23.279	23.458

Table 7.2.: Resulting confusion matrix for the experiment shown in Figure 7.1

7.1.2. Analysis of reasons for misclassification

In order to understand in which setting misclassifications occur, figure 7.2 shows some examples for misclassified frames. Each row shows one misclassified frame (in the middle), plus the frames before and after. Having a look at the False Negatives (Figure 7.2 (a)), we notice, that these images are all very similar in color and hence difficult to detect for the used algorithm. For example in (a1) the background is unchanged, as both interviews are recorded in front of the same wall. Furthermore the position of the microphones and faces overlap, so that the difference between both images does not exceed the given threshold.



(a) False Negatives (b) False Positives

Figure 7.2.: Examples for misclassified frames.

False Positives occurred in frames 8816*, 8817*, 13089*, 13090*, 13153*, 13154*, 14350*, 14351*, 14358*, 14359*, 14360*, 14363*, 14375*, 14382*, 14383*, 14385*, 14386*, 14390*, 14396*, 14480*, 14481*, 14493*, 14496*, 14506*, 14531*, 14532*, 14586*, 14710*, 14711*, 14733*, 14735*, 14736*, 14765*, 14766*, 14767*, 14768*, 15971, 15975, 15981, 17768, 20187, 20188, 20189, 20190, 20191, 20739*, 20740*, 20792*, 20793*, 20799* and 20800*. Three examples for False Positive classified frames are shown in Figure 7.2 (b).

Obviously, the misclassifications in the first and third example of figure 7.2 (b) occur due to overexposed frames. The reason for the occurrence of such frames are photographers using flashlights whilst the shot is recorded. These photographic flashlights cause large dissimilarity values of two adjacent frames, as every single pixel of the image get significant brighter. In the above given enumeration of all false positive frames from the "Tagesschau" show of September 11th, 2012, all frames that are misclassified due to photographic flashes are marked with the symbol *. As we see, photographic flashes are the

main reason for misclassification: 42 out of 51 misclassification occur due to photographic flashes. This phenomenon can be observed in the other news videos of the dataset as well. Hence it seems that photographic flashes are very common. Especially in TV news those flashes tend to produce a lot of segmentation errors and there are already existing approaches how to detect them (e.g. [Quénot et al., 2003], [Heng and Ngan, 2003]).

Flash detection is particularly interesting, as a high density of photographic flashes also is a good indicator for the importance of an events. They most often occur "in impressive scenes, such as interviews of important persons" [Takimoto et al., 2006]. Most likely scenes like that will be recorded by different cameras from different angles. In this scenario the pattern of photographic flashes can also used to identify the same scene in different videos [Takimoto et al., 2006]. Therefore photographic flashes can be a valuable semantic information and their identification might be an interesting task for the further ViSTA-TV project.

7.1.3. Real-time behavior of the approach

As figure 7.1 shows, the shot boundary detection experiment has been run on a scaled version of the news show video. This scaled version was used in order to speed the experiment up and reach a throughput of more than 25 frames per second. This throughput is necessary when working on live video data later on, as the television program delivered by Zattoo is broadcasted with 25 frames per second. As the throughput could also be increased by using better hardware, it nevertheless suggests itself to check, if the recognition rate improves when using video data with a higher resolution. Hence I have run the experiment on four different scaled versions of the video. The results of the experiments are shown in table 7.3.

video file (resolution)	**time needed** (in ms)	**FP**	**FN**
20120911-micro.raw (160 × 90)	67.797	49	3
20120911-small.raw (320 × 180)	193.935	51	3
20120911-scaled.raw (640 × 360)	583.756	51	3
20120911.raw (1280 × 720)	2.455.627	51	3

Table 7.3.: Quality of the classifier for different scaled versions of the news show video.

As we see, the experiment could not be accomplished in realtime using the video file with the maximum resolution. Taking 2.455.627 milliseconds ≈ 2.455 seconds ≈ 41 minutes to process and given that the video consists out of 23.457 frames, the throughput is less than 10 frames per second. Fortunately results on the scaled versions of the videos turned out to have the same quality than on the largest video. Hence processing scaled versions of the television programm will be suffient for the shot detection task in the ViSTA-TV project.

7.2. Segmentation and Tagging of News shows

7.2.1. Feature-based anchorshot detection using RapidMiner

As mentioned before, I decided to first of all extract a great amount of different features from the news show dataset by using RapidMiner and its Image Mining Extension. A full list of all extracted features can be found in the Appendix. In order to figure out, which of these features are useful, I trained a decision tree learner on the labeled dataset of three news shows. As described in chapter (see 3.2.2), decision tree learners work top-down by splitting the examples at each step using a feature that splits the example set best. Hence all features, that appear on one of the first levels of the inferred decision tree, have a high power to distinguish between the classes. For this experiment, I decided to focus on distinguishing between anchorshots and news report shots only. Thus the input for the learner consists out of a subset of 58.987 examples derives from the "Tagesschau" shows of September 13th and 15th, and October 27th 2012. The example set includes 14.249 anchorshots and 44.738 news report shots. The inferred decision tree is shown in figure 7.3.

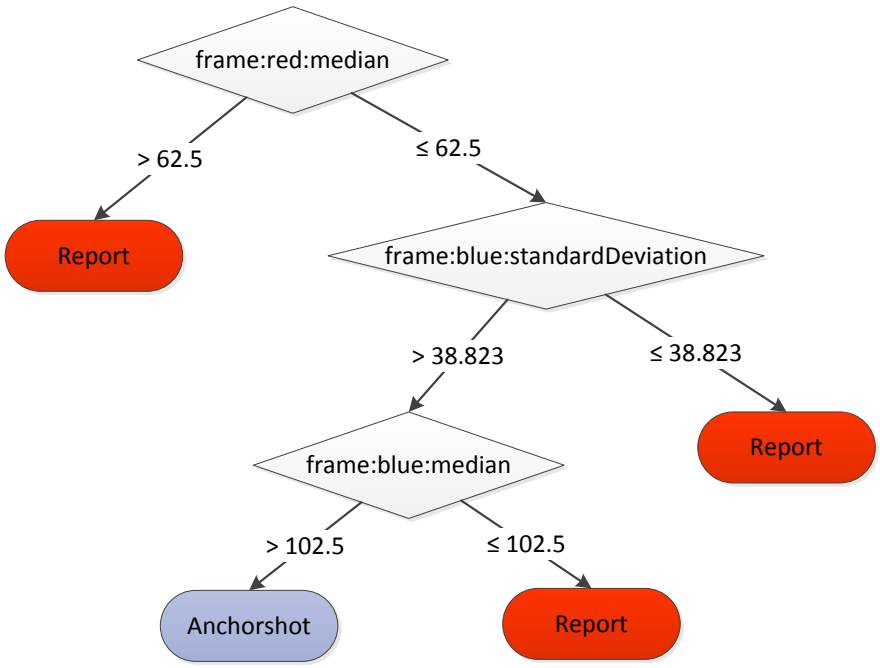


Figure 7.3.: Decision tree for the classification of anchorshots. The tree was inferred using all news report shots and anchorshots from three "Tagesschau" shows (Sep. 13th and 15th, and Oct. 27th 2012). Parameters for *Decision Tree* operator: {criterion = gain_ration, minimal size for split = 4, minimal leaf size = 2, minimal gain = 0.1, maximal depth = 4, confidence = 0.25, prepruning alternatives = 3}.

Obviously, the median red value, the standard deviation of the blue values, and the median blue value turn out to be features with good differentiation power between anchorshots and news report shots. When inferring a larger and deeper decision tree, further features like i.g. the normalized Y-coordinate of the center of mass of the red color channel proof to be useful as well. The value of these features also get emphasized when plotting the data. Figure 7.4 shows an example of such a plot. Obviously the median blue value (Y-axis) and the normalized Y-coordinate of the center of mass of the red color channel (X-axis) of anchorshots (within the given "Tagesschau" dataset) are always similar.

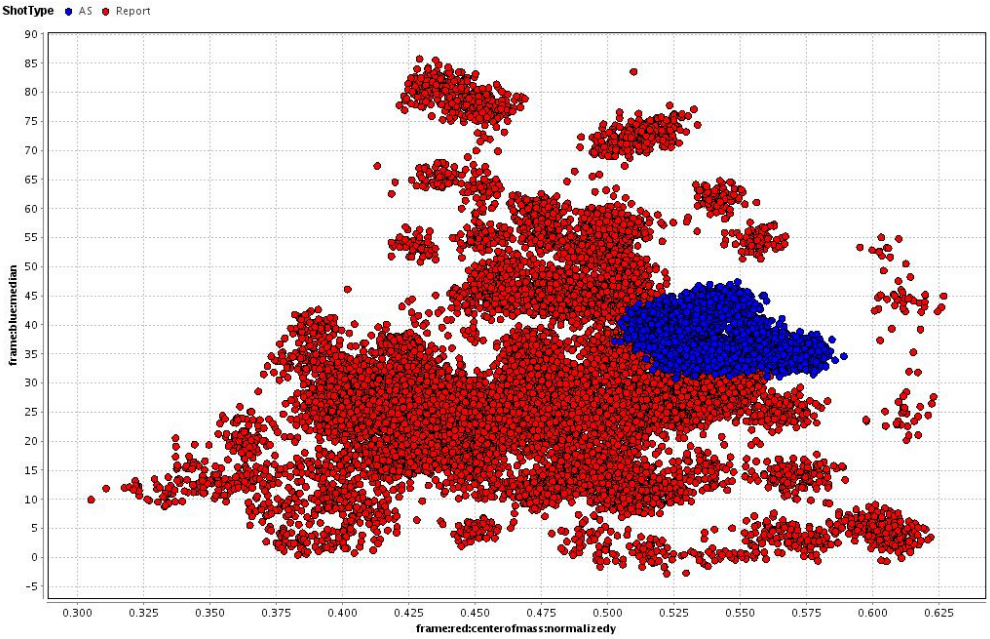


Figure 7.4.: The median blue value (Y-axis) and the normalized Y-coordinate of the center of mass of the red color channel (X-axis) have turned out to be features with good differentiation power between anchorshots (blue) and news report shots (red). This figure shows the corresponding values for 5000 randomly draw examples for news report shots and anchorshots from the "Tagesschau" show of September 11th, 2012. As the anchorshots are really similar and therefore overlaid each other, a small jitter has been applied for a better illustration.

Beside decision trees many other learning algorithms can be used to determine a set of features with good differentiation power. Support Vector Machines (SVM) assign weights to each feature. The higher the assigned weight for one feature is, the better this feature is to separate the classes. Hence I ran another experiment on the same data, using a linear SVM to build a model for anchorshot classification. The top five ranked features and their corresponding weights for two randomly drawn stratified samples of the data are listed in table 7.4.

feature name	**weight** (ranking) **in run1**	**weight** (ranking) **in run2**
frame:b_and_w:CenterOfMass:normalizedY	1.0 (1)	1.0 (1)
frame:red:standardDeviation	0.830 (2)	0.801 (3)
frame:blue:median	0.804 (3)	-
frame:green:average	0.789 (4)	-
frame:red:average	0.764 (5)	-
frame:blue:average	-	0.837 (2)
frame:green:CenterOfMass:normalizedX	-	0.729 (4)
frame:green:average	-	0.717 (5)

Table 7.4.: Normalized feature weights of a linear SVM for distinguishing between anchorshots and news report shots. The weights are calculated by using the *Weights by SVM* operator, which is included in RapidMiner. The calculation is based on two different randomly drawn stratified data sets, each including 10.000 examples.

Obviously, the optimal feature subset varies from one run to the other. This difference in the top five ranked features is based on the sampling of the input data. In order to have reliable results, we are interested in a more stable feature selection. Thus we have to find those features that produce good results on different splits of the data. Hence, after reading the labeled data in, we split the data into different sets, according to the split of an X-validation. As we will need another X-validation operator, this X-validation will be referred to as the outer X-validation. On each of the training data sets produced by the outer X-validation, we select the optimal features by using an evolutionary algorithm. This evolutionary algorithms again uses an (inner) X-validation to choose the selected features. Based on the optimal feature subset, a model is built and evaluated for each split of the original data. At the end, the outer X-validation computes, which features have been selected for most splits of the data. The corresponding process is shown in figure 7.5.

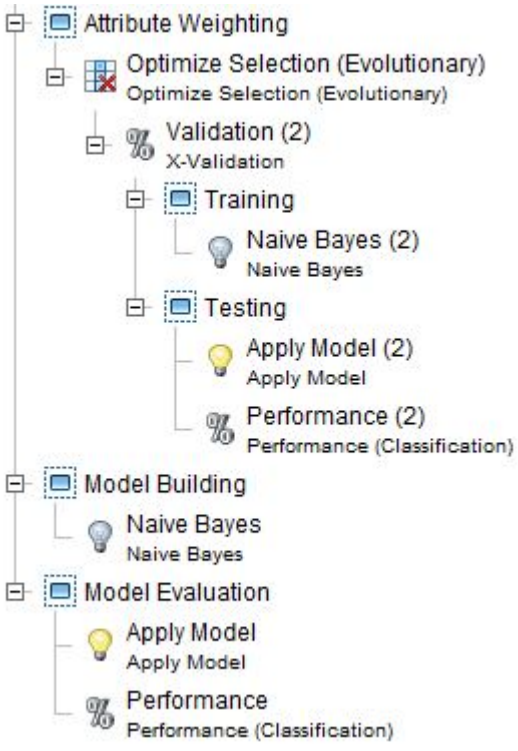


Figure 7.5.: RapidMiner process for the feature selection using Naive Bayes.

The reason for choosing an evolutionary feature selection is quite easy. As mentioned earlier, the idea was to extract as many features as possible by using RapidMiner IMMI. Hence the feature selection is performed on a large feature set. As the search space for a feature selection increases exponentially in the number of features (each feature can either be chosen or not), it is almost impossible to try each combination. The exponential feature selection operator tackles this problem.

Furthermore the evaluation of the selected features requires an inner X-validation. This leads to an enormous computational complexity, as the learning algorithms have to be trained and tested thousands of times. As training of an Support Vector Machine (SVM) is way more complex then the training of a simple learner like Naive Bayes, I decided to run the feature selection process, which is shown in figure 7.5, on Naive Bayes and Decision Trees only. Training the SVM has unfortunately turned out to be impossible in the available amount of time. Nevertheless, I expect these approaches to be a good first measure for the importance of the extracted features.

In theory, an inner parameter optimization for the learner would be needed as well. Due to the increasing computational complexity, I decided to rely on the default parameters for each learner in this setting. A parameter optimization is performed in another experiment, which is described later in this chapter.

Using Naive Bayes, "frame:red:average" and "frame:green:median" turned out to be the best features, both being selected in 90% of all (ten) splits of the data, performed by the outer X-validation. A full list of the top five ranked features is shown in table 7.5.

feature name	**weight**
frame:red:average	0.9
frame:green:median	0.9
frame:red:median	0.8
frame:blue:median	0.8
frame:b_a_w:CenterOfMass:normalizedX	0.7
frame:blue:average	0.7
frame:green:average	0.7
frame:red:standardDeviation	0.7

Table 7.5.: Top features selected for Naive Bayes for distinguishing between anchorshots and news report shots.

Of course the selected features do also strongly depend on the used learning algorithm. Hence it suggests itself to run the experiment for different learning algorithms. Table 7.6 shows an overview of the top five ranked features for different learning algorithms. In case more than one feature has the same weight than the fifth ranked feature, all equally ranked features are listed.

Learning Algorithm	**Selected Features** (weights)
Naive Bayes	frame:red:average (0.9), frame:green:median (0.9)
	frame:red:median (0.8), frame:blue:median (0.8)
	frame:b_a_w:CenterOfMass:normalizedX (0.7)
	frame:blue:average (0.7), frame:green:average (0.7)
	frame:red:standardDeviation (0.7)
Decision Tree	frame:red:median (1.0), frame:blue:median (0.9)
	frame:red:standardDeviation (0.9)
	frame:grey:CenterOfMass:normalizedX (0.8)
	frame:blue:average (0.7)
	frame:blue:CenterOfMass:normalizedX (0.7)
	frame:blue:CenterOfMass:normalizedY (0.7)
	frame:green:CenterOfMass:normalizedX (0.7)
	frame:red:CenterOfMass:normalizedY (0.7)

Table 7.6.: Top features selected by different learning algorithms for distinguishing between anchorshots and news report shots.

Beside a ranking of the features, the experiment also provides us with the accuracy of the used learning algorithms with regard to the learning task of detection anchorshots. The resulting accuracies are as follows: Naive Bayes scored 98.87% ± 0.19%. Decision Trees had a perfect performance of 100%.

7.2.2. Real-time model-based anchorshot detection

In chapter 2.3.3 I have presented three types of anchorshot detection approaches: model matching, face detection, and frame similarity. As mentioned, the frame similarity is an unsupervised approach clustering all shot and taking the most dense cluster as the anchorshot cluster. As said before, it is thus not easily transferable to stream data. Moreover, relying on face detection is only reasonable if position and size of the face are also taken into account. Hence the most promising and probably easiest approach for anchorshot detection on video streams is matching each shot against a model, which describes an anchorshot.

Applying a Decision Tree-Model

In the previous section 7.2.1, I have used RapidMiner to infer a decision tree that allows as to classify shots into anchorshots and news report shot. Although this model was learned offline, we can of course apply it online. This can be done by using the *ApplyDecisionTreeModel* processor. The corresponding process is shown in figure 7.6.

```
<container>

   <stream id="video"
      class="stream.io.MJpegImageStream"
      url="http://kirmes.cs.uni-dortmund.de/video/20120911-scaled.raw" />

   <process input="video" >
      <CreateID key="frame:id" />

      <stream.news.helper.AddNewsshowLabels />

      <stream.image.features.AverageRGB />
      <stream.image.features.MedianRGB />
      <stream.image.features.StandardDeviationRGB />

      <stream.news.learner.anchorshotdetection.ApplyDecisionTreeModel
            predictionkey="@prediction:anchorshot" />

      <stream.learner.evaluation.PredictionError label="@label:anchorshot" />

   </process>
</container>
```

Figure 7.6.: Stream Framework process to detect anchorshots by applying the decision tree inferred with RapidMiner.

The decision tree was inferred on the labeled "Tagesschau" news videos from September 13th and 15th, and October 27th, 2012. Hence it makes sense to apply it on the "Tagesschau" news video broadcasted on September 11th, 2012. Based on this news video, the precision of the detection approach reaches 97% (recall = 85.8%). Unfortunately only half of the real anchorshots really get labeled as anchorshots. This is of course not sufficient, but can be improved by using a decision tree, which is deeper and hence takes into account more features.

	Prediction: true	Prediction: false	Total
Label: True	2.966	2.817	5.783
Label: False	529	17.149	17.678
Total	3.495	19.966	23.461

Table 7.7.: Resulting confusion matrix for the anchorshot detection experiment shown in figure 7.6

Applying an Image-Model

But not only decision trees are reasonable models for representing anchorshots. Anchorshots can also be represented by images. I have considered two types of images.

Inferred anchorshot mask

Based on labeled data, we can evaluate, which pixels in anchorshot frames never change at all. This includes all segments of anchorshot frames, which belong to the studio

furnishings or the background. The advantage of this model is that it can be updated anytime a new anchorshot is found by simply comparing the new anchorshot to the old model and exclude pixels that differ. Figure 7.7 shows an example for the resulting mask. Excluded pixels are represented as black pixels, whereas all colored pixels are pixels that had this color (± a limit of tolerance) throughout all anchorshot frames seen so far. The presented mask has been inferred automatically by using the *AnchorshotModelCreator* processor, I have implemented for the streams framework.

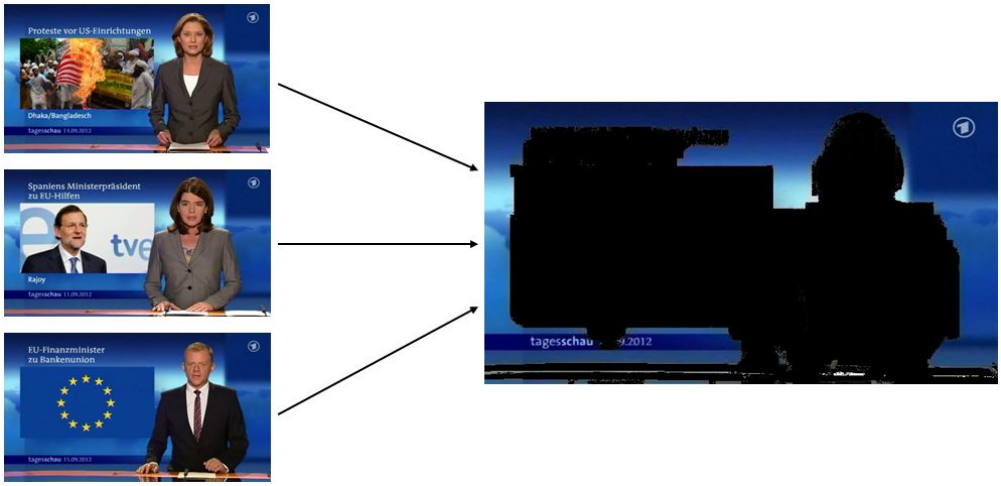


Figure 7.7.: Automatically inferred model for anchorshots using the labeled data included in the news dataset.

First anchorshot frame of the show

We know that each "Tagesschau" show starts with an intro, directly followed by an anchorshot. As the intro of each "Tagesschau" show is always the same and covers approximately 250 frames, we can assume, that the 300th frame of a "Tagesschau" news show is an anchorshot. Hence we can compare each following frame to the 300th frame and compute the pixel-wise color difference. If at least 50% of the pixels are the same (± a limit of tolerance), I classify the frame as an anchorshot frame. Surprisingly this easy approach turns out to perform very well, resulting in an accuracy of 100%. As table 7.8 shows, there were no misclassifications at all when applying this on the "Tagesschau" news video from September 11th, 2012.

	Prediction: true	Prediction: false	Total
Label: True	5.783	0	5.783
Label: False	0	17.678	17.678
Total	5.783	17.678	23.461

Table 7.8.: Resulting confusion matrix for the anchorshot detection experiment using the pixel-wise comparision of each frame with the 300th frame for the show.

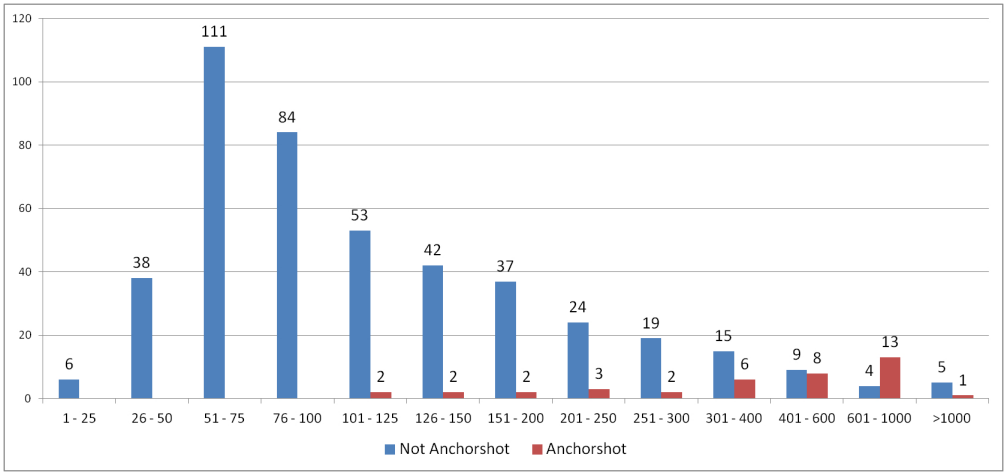


Figure 7.8.: The number of frames for each shot within the "Tagesschau" news dataset, distinguishing between anchorshots and 'normal' shots.

Computational time optimization

Of course, it is not necessary to perform the anchorshot detection for each single frame. In fact it is sufficient to test one frame per shot only. This enables us to significantly speed the anchorshot detection process up. Furthermore, when looking at the data included in the news dataset, it turns out that anchorshots are significantly longer than news report shots. Table 7.9 shows this aspect by comparing the average number of frames of an anchorshot and the average number of frames of a news report shot.

Date	# of anchorshots	**average # of frames**	# of other shots	**average # of frames**
11.09.2012	11	412	129	138
13.09.2012	12	520	121	144
15.09.2012	12	383	96	144
27.10.2012	10	508	105	146
Total	48	452	450	143

Table 7.9.: Number of frames in anchorshots vs. other shots in "Tagesschau" news shows.

In fact all anchorshots in the news dataset cover more than 100 frames. Hence it is very unlikely, that short shots, covering less than four seconds (≈ 100 frames), are anchorshots. Figure 7.8) illustrates this fact. Utilizing this fact as well, the computational time of the anchorshot detection could be improved by the factor 100.

7.3. Coffee capsule recognition

The second use case is about detecting and recognizing coffee capsules in video data. It splits in two task: the event detection and capsule recognition. Both problems are tackled in this chapter.

7.3.1. Event Detection

Figure 5.11 has already given us an idea, how the frames that belong to one event look like. Obviously the images of all frames, which do not belong to an event, do not differ too much. But whenever a coffee capsule slips by, the color of the capsule changes the image extremely. Hence I started my experiments by calculating the average color values of all three RGB color channels for each image and plotted the data, using the `stream.plotter.Plotter` processor, which comes with the streams framework. The resulting plot is shown in figure 7.9.

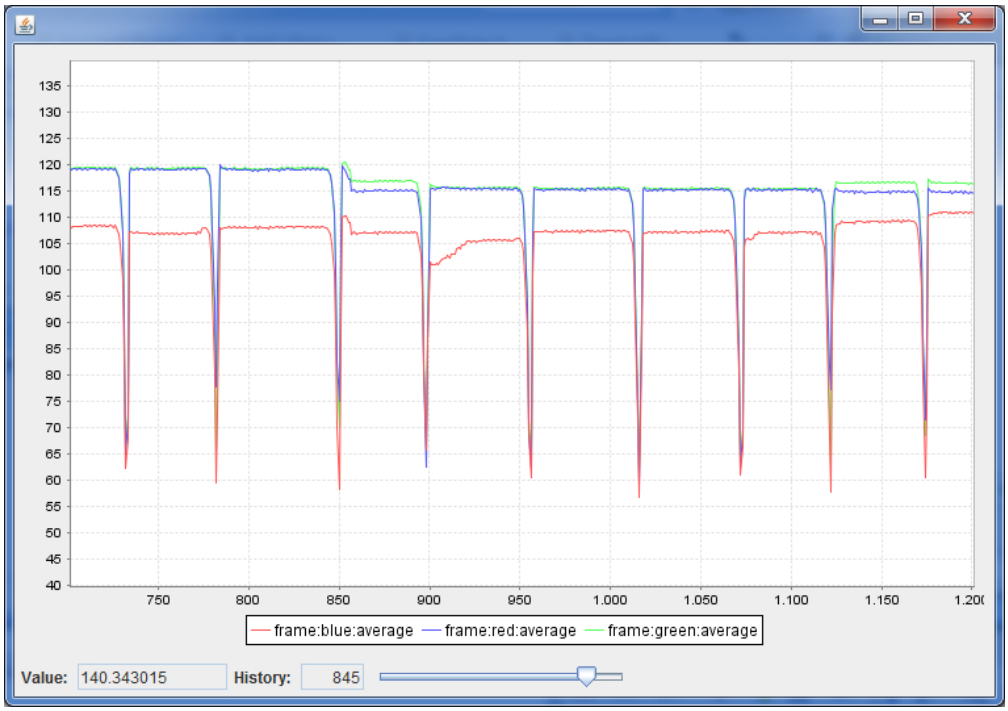


Figure 7.9.: Average RGB values for capsule events.

The plot visualizes the average RGB color values for a sequence of 500 frames, taken from one of the video files included in the "coffee"-dataset. The sequence covers 9 events of coffee capsules slipping down the slide. Obviously, the assumption that the average RGB color values are a good indicator for recognizing events, is supported: as long as no capsule slips by, the average RGB color values of the frames do not change a lot. The average blue value (in this plot visualized by the red line) is somewhere around 110 ± 10, the average red and green color values are somewhere around 120 ± 10. As soon as the frame shows a coffee capsule, the average color values for all three RGB

channels decrease significantly. In this sequence they all drop clearly below 90. Hence my first approach for detecting events in the "coffee"-dataset bases on the extraction of the average color values for all RGB color channels. Based on this, a simple classifier tests, whether the value for one of the color channels dips more than a fixed threshold t, or not. In case it does, the frame is classified as "belongs to an event". The corresponding configuration file for running this with the streams framework is shown in figure 7.10.

```
<container>

   <stream id="video"
      class="stream.io.MJpegImageStream"
      url="http://mattis.special-operations.de/video/kapseln.raw" />

   <process input="video" >
      <CreateID key="frame:id" />
      <Skip condition="\%{data.frame:id} @lt 30" />

      <stream.image.DisplayImage image="data" />

      <stream.coffee.helper.AddCoffeeLabels
            file="file:///C:/ (....) /data/kapseln/kapseln.csv" />

      <stream.image.features.AverageRGB />

      <stream.coffee.eventdetection.ThresholdEventDetection
            predictionkey="@prediction:event" attribute="frame:red:average"
            standardvalue="120" t="15" />

      <stream.plotter.Plotter history="500" keepOpen="true"
         keys="frame:red:average,frame:blue:average,frame:green:average" />

      <stream.coffee.eventdetection.EventDetectionEvaluation />
   </process>
</container>
```

Figure 7.10.: Streams Framework process to detect events in one of the coffee capsule video file by applying the simple classifier using a threshold.

As already mentioned in chapter 5.2.2, an event ranges over more than one frame. Nevertheless it does not matter, if the event detection algorithms does not label all of the frames, belonging to the event, as such. In fact it is sufficient, when at least one of the frames is detected. Hence I needed a new evaluation processor to determine the quality of the classifier. This new processor is named `stream.coffee.eventdetection.EventDetectionEvaluation` processor and further described in the appendix.

Running the experiment, it turns out that this easy classifier is sufficient to solve the learning task. On both video data files included in the dataset, it detects all events, without ever proclaiming an event, where no event is. Hence, precision as well as recall for this classifier are 100% on the given data. Of course the "standardvalue" and the "threshold" used by the classifier will have to be adjusted to the data, produced by a web cam, which is installed on the coffee maker directly.

7.3.2. Capsule Recognition

After having detected the events, the next task is to classify the color of the coffee capsules correctly. When zooming in the dips, which can be observed as soon as a coffee capsule slips by, I noticed that the dips look slightly different based on the color the coffee capsule had. Figure 7.11 illustrates this. Hence I decided to determine the minimal observed color value for each color channel during one event. Afterwards I stored these values into a CSV file, including the real label of the event. I did that for both video sequences included in the dataset. Thus I got 72 examples, each containing three numerical values plus a label. This is then taken as the input for RapidMiner.

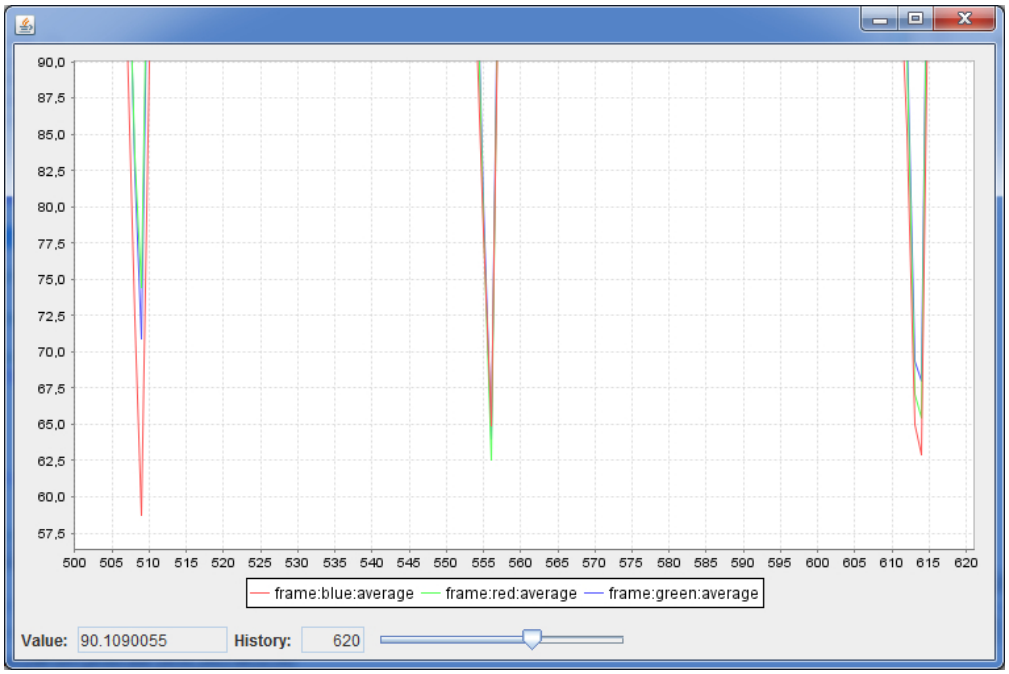
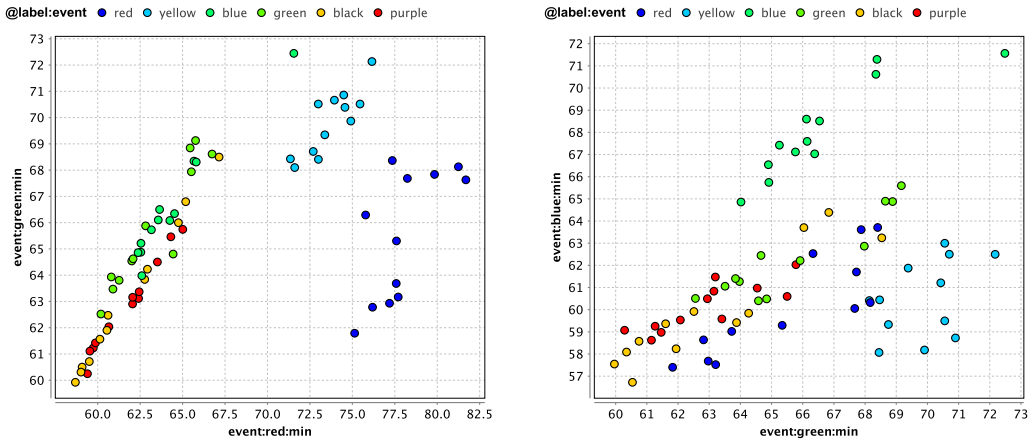


Figure 7.11.: Average RGB values for different colored coffee capsules. This plot is just a zoomed-in view of the plot, shown in figure 7.9.

Graphical analysis of the data

When plotting the data by choosing two of the three color channels, the minimal RGB values have been evaluated for, and indicating the label by the color of the points, it turns out, that the minimal observed color values for the three color channels during one event really seem to be useful features with good differentiation power. Figure 7.12 shows two of the three possible plots. Obviously yellow, red, and green capsules are best recognizable, as they form the most dense clusters in the plots.

As we are having labeled data and the provided features seem to be useful, we can now start to train and evaluate possible models.



(a) red and green channel

(b) blue and green channel

Figure 7.12.: Min RGB values for different colored coffee capsules.

Inferring a model and evaluation

Based on all 72 events, included in the two datasets, I now try to develop a model by using different machine learning algorithms. The RapidMiner process for inferring and testing a model is shown in figure 7.13. It consists of two *Read CVS* operators, which read in the data produced by the streams framework. After joining the data, it gets normalized to a range from 0 to 1. Afterwards it is splitting into ten sets of training and testing data by using a X-validation. The model is inferred on nine of the data sets, whilst one set is held back for testing. The parameters of the learner are optimized by using an evolutionary parameter optimization.
As the full data set only covers 72 examples of coffee capsules events, the overall process including X-validation and parameter optimization can be run in a reasonable amount of time for different machine learning algorithms. The best performance was obtained by a LibSVM (C-SVC, linear kernel, C=6233.8, epsilon 0.0010), classifying the data with an accuracy of 86.25 % ± 8.53 %. The resulting confusion matrix is shown in figure 7.10.

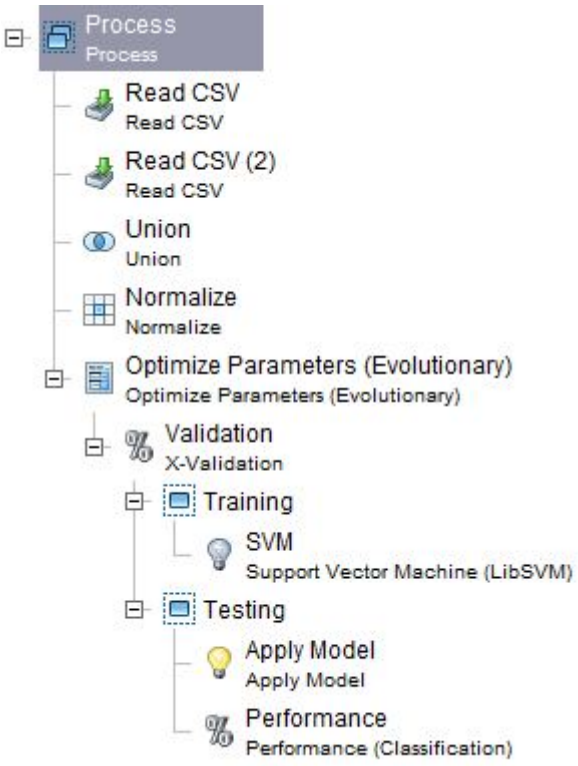


Figure 7.13.: RapidMiner process for inferring and testing a classifier on the extracted dataset.

	Pred: red	Pred: yellow	Pred: blue	Pred: green	Pred: black	Pred: purple	class recall
Label: red	12	0	0	0	0	0	100 %
Label: yellow	0	12	0	0	0	0	100 %
Label: blue	0	0	12	0	0	0	100 %
Label: green	0	0	0	11	0	1	91.7 %
Label: black	0	0	0	0	10	2	83.3 %
Label: purple	0	0	0	0	5	7	58.3 %
class precision	100 %	100 %	100%	100 %	66.7 %	70.0 %	

Table 7.10.: Confusion matrix for the capsule color detection using LibSVM.

Beside the LibSVM, I have tested other learning algorithms. The classification results can be found in table 7.11.

Obviously the detection of certain capsules is quite easy. Red, yellow and blue capsules get recognized easily, the predictions for green are sufficient as well. Only the black and the purple capsules can hardly be distinguished. I believe, that the reason for this is the poor quality and the bad light circumstances of the video data. When pointing a spot light on the capsules, at the time they are filmed, the classification will surely become better. Nevertheless an accuracy of more than 85% is quite good in a setting, with six equally distributed classes. Random guessing would after all result in less than 20% accuracy.

algorithm	**accuracy**	optimal parameters
k-NN	79.46 % ± 15.18 %	k=1
Naive Bayes	76.86 % ± 11.52 %	
Neural Net	76.43 % ± 10.97 %	training cycles = 63, learning rate = 0.863, momentum = 0.564
Decision Tree	63.39 % ± 16.39 %	criterion = gain ratio, minimal gain = 0.05, minimal leaf size = 1, minimal size for split = 3

Table 7.11.: Results for the coffee capsule recognition using different machine learning algorithms.

8

Outlook and Conclusion

The experiments have shown that the implemented methods for video segmentation and tagging perform well on the presented datasets. Nevertheless there is a lot that could be done in future to obtain good results on random video data.

Outlook

The problem of detecting shot boundaries can be tackled by detecting cuts and gradual transitions. The presented approaches are capable to handle almost any kind of video data. Minor errors due to slow gradual transitions, which can hardly be detected, are bearable, as cuts are by far the more common type of shot transitions. Nevertheless "it reveals that the identification of cuts has been somewhat successfully tackled, while the detection of gradual transitions still remains a difficult problem" [Yuan et al., 2007], which should be addressed in future. Furthermore, the experiments have shown that it would be useful to be able to distinguish between cuts and noises like photographic flashlights.

More data

Segment detection is far more difficult on random video data. It turned out that the problem can be tackled by taking into account prior knowledge about the type of video. That way, I have successfully been able to segment the "Tagesschau" news videos included in the "news" dataset, by identifying sets of features with good differentiation power. Although the set of valuable features will surely vary from one news show to the other, the method of segmenting news by detecting anchorshots can be done for other news formats as well, using the same techniques. Similar approaches do also exist for various other types of video data. Nevertheless, there is still work to be done in order to obtain a good automatic segmentation and tagging on random video data. This problem will hopefully be tackled successfully during the next steps of the ViSTA-TV project.

More features

Beside considering a wider field of video content, it would be helpful to extract even more features from the incoming video data. By extracting features directly from MPEG compressed video data, things could further been fastened up. Additionally some scientists have already proven that MPEG features can be useful to detect video segments ([Lee et al., 2000]). Furthermore it turned out that including audio data, as long as it comes along with the video data, can be useful as well. Daxenberger's approach for audio segmentation [Daxenberger, 2007] shows, how good segmentation on audio data can be solely. Using multi-modal approaches, as described in chapter 2.5.1, will even improve the segmentation quality. Finally, EPG data would also deliver a bunch of new and interesting features.

When the amount of extractable features rises, the selection of valuable features has to be repeated as well. Beside focusing on features that get selected by certain machine learning algorithms, a direct feature selection is also possible using the RapidMiner feature selection extension by Schowe [Schowe, 2011]. This could also be tried out in future.

More possible tags

Last but not least, the presented use cases do not at all cover all possible tags that would be useful within the ViSTA-TV project. As mentioned in chapter 2.4, a further emphasis should be put on the extraction of more tags. Good examples are "advertise" vs. "no advertise", "photographic flashlight", or the names of actors.

Conclusion

This work shows that a huge amount of reasonable features for video segmentation and tagging can be extracted on live data in real-time. By applying the extracted features on two use cases, I have shown that an adequate segmentation based on these features is possible. Furthermore, the presented process for feature extraction and selection can easily be adapted to new features. The streams framework hereby helps to implement new processors, which are capable to extract features in a real-time stream setting. Examples for more features or tags, which at the end are features as well, are mentioned above and described in the second chapter of this thesis.

Bibliography

[itu, 2002] (2002). *ITU-R BT.709: Parameter values for the HDTV standards for production and international programme exchange*. International Telecommunication Union.

[Aggarwal et al., 2003] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB '03)*, volume 29 of *VLDB '03*, pages 81–92. VLDB Endowment.

[Ahanger and Little, 1996] Ahanger, G. and Little, T. D. C. (1996). A survey of technologies for parsing and indexing digital video. *Journal of visual Communication and image representation*, 7:28–43.

[Aigrain and Joly, 1994] Aigrain, P. and Joly, P. (1994). The automatic real-time analysis of film editing and transition effects and its applications. *Computers and Graphics*, 18(1):93 – 103.

[Aigrain et al., 1997] Aigrain, P., Joly, P., and Longueville, V. (1997). Medium knowledge-based macro-segmentation of video into sequences. In Maybury, M. T., editor, *Intelligent multimedia information retrieval*, chapter Medium knowledge-based macro-segmentation of video into sequences, pages 159–173. MIT Press, Cambridge, MA, USA.

[Alvarez et al., 1992] Alvarez, L., Lions, P.-L., and Morel, J.-M. (1992). Image selective smoothing and edge detection by nonlinear diffusion. ii. *SIAM Journal on Numerical Analysis*, 29(3):845–866.

[Arasu and Manku, 2004] Arasu, A. and Manku, G. S. (2004). Approximate counts and quantiles over sliding windows. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems*, PODS '04, pages 286–296, New York, NY, USA. ACM.

[Arman et al., 1993] Arman, F., Hsu, A., and yee Chiu, M. (1993). Feature management for large video databases. In *Proceedings of SPIE Storage and Retrieval for Image and Video Databases*, pages 2–12.

[Avrithis et al., 2000] Avrithis, Y., Tsapatsoulis, N., and Kollias, S. (2000). Broadcast news parsing using visual cues: A robust face detection approach. In *Proceedings of*

IEEE International Conference on Multimedia and Expo (ICME 2000), New York City, NY, USA.

[Babaguchi et al., 2004] Babaguchi, N., Kawai, Y., Ogura, T., and Kitahashi, T. (2004). Personalized abstraction of broadcasted american football video by highlight selection. *IEEE Transactions on Multimedia*, 6(4):575 – 586.

[Balke et al., 2010] Balke, M., Beckers, T., Fernys, K., Haak, D., Homburg, H., Kalabis, L., Kokott, M., Kulmann, B., Müller, K., Przyluczky, C., Schulte, M., Skirzynski, M., Bockermann, C., and Schowe, B. (2010). Endbericht Projektgruppe 542: Stream Mining for Intrusion Detection in Distributed Systems. Technical report, TU Dortmund University. Previous number = SIDL-WP-1999-0120.

[Bertini et al., 2001] Bertini, M., Del Bimbo, A., and Pala, P. (2001). Content-based indexing and retrieval of tv news. *Pattern Recognition Letters*, 22(5):503–516.

[Bhabatosh Chanda, 2011] Bhabatosh Chanda, D. (2011). *Digital Image processing and analysis*. PHI Learning Private Limited.

[Bockermann and Blom, 2012a] Bockermann, C. and Blom, H. (2012a). Processing Data Streams with the RapidMiner Streams-Plugin. In *Proceedings of the 3rd RapidMiner Community Meeting and Conference*.

[Bockermann and Blom, 2012b] Bockermann, C. and Blom, H. (2012b). The streams framework. Technical Report 5, TU Dortmund University.

[Bordwell and Thompson, 2012] Bordwell, D. and Thompson, K. (2012). *Film Art: An Introduction*. Film Art: An Introduction. McGraw-Hill College.

[Boreczky and Rowe, 1996] Boreczky, J. S. and Rowe, L. A. (1996). Comparison of video shot boundary detection techniques. In *Journal of Electronic Imaging*, volume 5, pages 122 – 128.

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

[Browne et al., 1999] Browne, P., Smeaton, A. F., Murphy, N., O'Connor, N., Marlow, S., and Berrut, C. (1999). Evaluating and combining digital video shot boundary detection algorithms. In *In Processing of the Irish Machine Vision and Image Conference (IMVIP 2000)*.

[Brunelli et al., 1996] Brunelli, R., Mich, O., and Modena, C. M. (1996). A survey on video indexing. *Journal of Visual Communications and Image Representation*, 10:78–112.

[Burger and Burge, 2007] Burger, W. and Burge, M. (2007). *Digital Image Processing: An Algorithmic Introduction Using Java*. Texts in Computer Science. Springer.

[Burget et al., 2011] Burget, R., Cika, P., Zukal, M., and Masek, J. (2011). Automated localization of temporomandibular joint disc in mri images. In *Proceedings of the 34th International Conference on Telecommunications and Signal Processing (TSP 2011)*, pages 413 –416.

[Burget et al., 2010] Burget, R., Karásek, J., Smékal, Z., Uher, V., and Dostál, O. (2010). Rapidminer image processing extension: A platform for collaborative research. *The 33rd International Conference on Telecommunication and Signal Processing, TSP*, 2010:114–118.

[Chang et al., 2004] Chang, S.-L., Chen, L.-S., Chung, Y.-C., and Chen, S.-W. (2004). Automatic license plate recognition. *IEEE Transactions on Intelligent Transportation Systems*, 5(1):42 – 53.

[Charikar et al., 2004] Charikar, M., Chen, K., and Farach-Colton, M. (2004). Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15.

[Chen and Tu, 2007] Chen, Y. and Tu, L. (2007). Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 133–142, New York, NY, USA. ACM.

[Cisco, 2012] Cisco (2012). Visual networking index: Forecast 2011-2016.

[Cotsaces et al., 2006] Cotsaces, C., Nikolaidis, N., and Pitas, I. (2006). Video shot detection and condensed representation. a review. *Signal Processing Magazine, IEEE*, 23(2):28 –37.

[Dailianas et al., 1995] Dailianas, A., Allen, R. B., and England, P. (1995). Comparison of automatic video segmentation algorithms. In *SPIE Photonics West*, pages 2–16.

[Daxenberger, 2007] Daxenberger, A. (2007). Datengestützte Segmentierung von Audiodaten.

[De Santo et al., 2007] De Santo, M., Percannella, G., Sansone, C., and Vento, M. (2007). Segmentation of news videos based on audio-video information. *Pattern Analysis & Applications*, 10:135–145. 10.1007/s10044-006-0055-5.

[Domingos and Hulten, 2000] Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA. ACM.

[Dries and Rückert, 2009] Dries, A. and Rückert, U. (2009). Adaptive concept drift detection. *Stat. Anal. Data Min.*, 2(5):311–327.

[Farabet et al., 2011] Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., and LeCun, Y. (2011). NeuFlow: A runtime reconfigurable dataflow processor for vision. In *Proceedings of the 5th IEEE Workshop on Embedded Computer Vision.*

[Fayyad et al., 1996] Fayyad, U., Piatetsky-shapiro, G., Smyth, P., and Widener, T. (1996). The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39:27–34.

[Gaber et al., 2005] Gaber, M. M., Zaslavsky, A., and Krishnaswamy, S. (2005). Mining data streams: a review. *SIGMOD Record*, 34(2):18–26.

[Gao and Tang, 2002] Gao, X. and Tang, X. (2002). Unsupervised video-shot segmentation and model-free anchorperson detection for news video story parsing. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(9):765 – 776.

[Gargi et al., 2000] Gargi, U., Kasturi, R., and Strayer, S. (2000). Performance characterization of video-shot-change detection methods. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(1):1 –13.

[Gunsel et al., 1996] Gunsel, B., Ferman, A. M., and Tekalp, A. M. (1996). Video indexing through integration of syntactic and semantic features. In *Proceedings of the 3rd IEEE Workshop on Applications of Computer Vision (WACV '96)*, WACV '96, pages 90–, Washington, DC, USA. IEEE Computer Society.

[Han and Kamber, 2006] Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems Series. Elsevier Science & Technology.

[Hanjalic, 2002] Hanjalic, A. (2002). Shot-boundary detection: unraveled and resolved? *IEEE Transactions on Circuits and Systems for Video Technology*, 12(2):90 –105.

[Hanjalic et al., 1999] Hanjalic, A., Lagendijk, R., and Biemond, J. (1999). Semi-automatic news analysis, indexing, and classification system based on topics preselection. In *Proceedings of SPIE: Electronic Imaging: Storage and Retrieval of Image and Video Databases*.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[Heng and Ngan, 2003] Heng, W. J. and Ngan, K. N. (2003). High accuracy flashlight scene determination for shot boundary detection. *Signal Processing: Image Communication*, 18(3):203 – 219.

[Ide et al., 1998] Ide, I., Yamamoto, K., and Tanaka, H. (1998). Automatic video indexing based on shot classification. In *Proceedings of the 1st International Conference on Advanced Multimedia Content*, pages 87–102. Springer-Verlag.

[Kasturi and Jain, 1991] Kasturi, R. and Jain, R. (1991). *Computer Vision: Principles*. IEEE catalog. IEEE Computer Society Press.

[Koprinska and Carrato, 2001] Koprinska, I. and Carrato, S. (2001). Temporal video segmentation: A survey captions on mpeg compressed video. *DSignal Processing Image Communication*, pages 477–500.

[Kraaij et al., 2005] Kraaij, W., Smeaton, A. F., Over, P., and Arlandis, J. (2005). Trecvid 2004 - an overview. In *Online Proceedings of TREC Video Retrieval Evaluation*.

[Lee et al., 2000] Lee, S.-W., Kim, Y.-M., and Choi, S. W. (2000). Fast scene change detection using direct feature extraction from mpeg compressed videos. *IEEE Transactions on Multimedia*, 2(4):240–254.

[Lefevre et al., 2003] Lefevre, S., Holler, J., and Vincent, N. (2003). A review of real-time segmentation of uncompressed video sequences for content-based search and retrieval. *Real-Time Imaging*, 9(1):73 – 98.

[Li and Ngan, 2011] Li, H. and Ngan, K. (2011). Image/video segmentation: Current status, trends, and challenges. In Ngan, K. N. and Li, H., editors, *Video Segmentation and Its Applications*, pages 1–23. Springer New York.

[Lienhart, 1999] Lienhart, R. (1999). Comparison of automatic shot boundary detection algorithms. In *Proceedings of SPIE*, volume 3656, pages 290–301.

[Lienhart et al., 1997a] Lienhart, R., Kuhmunch, C., and Effelsberg, W. (1997a). On the detection and recognition of television commercials. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems '97*, pages 509 –516.

[Lienhart et al., 1997b] Lienhart, R., Pfeiffer, S., and Effelsberg, W. (1997b). Video abstracting. *Commun. ACM*, 40(12):54–62.

[Liu et al., 1998] Liu, Z., Wang, Y., and Chen, T. (1998). Audio feature extraction and analysis for scene segmentation and classification. *The Journal of VLSI Signal Processing*, 20:61–79.

[MacQueen, 1967] MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In Cam, L. M. L. and Neyman, J., editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press.

[Manku and Motwani, 2002] Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002)*, VLDB '02, pages 346–357. VLDB Endowment.

[McGee et al., 2005] McGee, T., Sengupta, R., and Hedrick, K. (2005). Obstacle detection for small autonomous aircraft using sky segmentation. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA 2005)*, pages 4679 – 4684.

[Merlino et al., 1997] Merlino, A., Morey, D., and Maybury, M. (1997). Broadcast news navigation using story segmentation. In *Proceedings of the 5th ACM International Conference on Multimedia*, pages 381–391.

[Mierswa and Morik, 2005] Mierswa, I. and Morik, K. (2005). Automatic feature extraction for classifying audio data. *Machine Learning Journal*, 58:127–149.

[Mierswa et al., 2006] Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In Ungar, L., Craven, M., Gunopulos, D., and Eliassi-Rad, T., editors, *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06)*, pages 935–940, New York, NY, USA. ACM.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New York.

[Nagasaka and Tanaka, 1992] Nagasaka, A. and Tanaka, Y. (1992). Automatic video indexing and full-video search for object appearances. In *Proceedings of the IFIP TC2/WG 2.6 Second Working Conference on Visual Database Systems*, pages 113–127, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co.

[O'Callaghan et al., 2002] O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S., and Motwani, R. (2002). Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering*, pages 685 –694.

[onlinekosten.de GmbH, 2012] onlinekosten.de GmbH (2012). Zattoo wächst auf 8 Millionen Nutzer. zuletzt abgerufen am 20.06.2012.

[Pantos and May, 2009] Pantos, E. R. and May, W. (2009). Internet draft - http live streaming.

[Patel and Sethi, 1996] Patel, N. and Sethi, I. (1996). Compressed video processing for cut detection. *IEEE Proceedings on Vision, Image and Signal Processing*, 143(5):315 –323.

[Pomerleau, 1993] Pomerleau, D. A. (1993). *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic Publishers, Norwell, MA, USA.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning Journal*, 1(1):81–106.

[Quénot et al., 2003] Quénot, G., Moraru, D., and Besacier, L. (2003). Clips at trecvid: Shot boundary detection and feature detection. In *Proceedings of TRECVID 2003 Workshop Notebook Papers*, pages 35–40.

[Rafael C Gonzalez and Woods, 2008] Rafael C Gonzalez, P. and Woods, R. (2008). *Digital Image Processing, Third Edition*. Prentice Hall.

[Rahman and Kehtarnavaz, 2008] Rahman, M. and Kehtarnavaz, N. (2008). Real-time face-priority auto focus for digital and cell-phone cameras. *IEEE Transactions on Consumer Electronics*, 54(4).

[Richardson, 2003] Richardson, I. E. G. (2003). *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*. John Wiley & Sons, Inc., New York, NY, USA.

[Santo et al., 2004] Santo, M., Percannella, G., Sansone, C., and Vento, M. (2004). Combining experts for anchorperson shot detection in news videos. *Pattern Analysis and Applications*, 7:447–460.

[Sato et al., 1999] Sato, T., Kanade, T., Hughes, E. K., Smith, M. A., and Satoh, S. (1999). Video ocr: indexing digital news libraries by recognition of superimposed captions. *Multimedia Systems*, 7:385–395.

[Schowe, 2011] Schowe, B. (2011). Feature Selection for high-dimensional data in RapidMiner. In *Proceedings of the 2nd RapidMiner Community Meeting And Conference (RCOMM 2011)*. Shaker Verlag.

[Shearer, 2000] Shearer, C. (2000). The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4).

[Smeaton et al., 2006] Smeaton, A. F., Over, P., and Kraaij, W. (2006). Evaluation campaigns and trecvid. In *Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval (MIR '06)*, pages 321–330, New York, NY, USA. ACM Press.

[Smoliar and Zhang, 1994] Smoliar, S. W. and Zhang, H. (1994). Content-based video indexing and retrieval. *IEEE MultiMedia*, 1(2):62–72.

[Snoek and Worring, 2003a] Snoek, C. G. and Worring, M. (2003a). Time interval maximum entropy based event indexing in soccer video. In *IEEE International Conference on Multimedia & Expo (ICME 2013)*, page pp.

[Snoek and Worring, 2003b] Snoek, C. G. M. and Worring, M. (2003b). Goalgle: A soccer video search engine. In *IEEE International Conference on Multimedia & Expo (ICME 2013)*.

[Snoek and Worring, 2005] Snoek, C. G. M. and Worring, M. (2005). Multimodal video indexing: A review of the state-of-the-art. *Multimedia Tools and Applications*, 25(1):5–35.

[Stringa and Regazzoni, 2000] Stringa, E. and Regazzoni, C. (2000). Real-time video-shot detection for scene surveillance applications. *IEEE Transactions on Image Processing*, 9(1):69–79.

[Strutz and Strutz, 2009] Strutz, T. and Strutz, T. (2009). Grundlegende verfahren zur bildsequenzkompression. In *Bilddatenkompression*, pages 261–272. Vieweg+Teubner.

[Sundaram and Chang, 2000] Sundaram, H. and Chang, S.-F. (2000). Video scene segmentation using video and audio features. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2000)*, volume 2, pages 1145 –1148.

[Swain and Ballard, 1991] Swain, M. J. and Ballard, D. H. (1991). Color indexing. *International Journal of Computer Vision (IJCV)*, 7(1):11–32.

[Swanberg et al., 1993] Swanberg, D., Shu, C.-F., and Jain, R. C. (1993). Knowledge-guided parsing in video databases. *Proceedings of SPIE Storage and Retrieval for Image and Video Databases*, pages 13–24.

[Takimoto et al., 2006] Takimoto, M., Satoh, S., and Sakauchi, M. (2006). Identification and detection of the same scene based on flash light patterns. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2006)*, pages 9 –12.

[Tonomura and Abe, 1989] Tonomura, Y. and Abe, S. (1989). Content oriented visual interface using video icons for visual database systems. In *IEEE Workshop on Visual Languages, 1989*, pages 68 –73.

[Turk et al., 1987] Turk, M., Morgenthaler, D., Gremban, K., and Marra, M. (1987). Video road-following for the autonomous land vehicle. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, volume 4, pages 273–280.

[Uher and Burget, 2012] Uher, V. and Burget, R. (2012). Automatic 3d segmentation of human brain images using data-mining techniques. In *Proceedings of the 35th International Conference on Telecommunications and Signal Processing (TSP 2012)*, pages 578 –580.

[Vaiapury and Izquierdo, 2010] Vaiapury, K. and Izquierdo, E. (2010). An o-fdp framework in 3d model based reconstruction. In *APWeb*, pages 424–429.

[Wang et al., 2000] Wang, Y., Liu, Z., and Huang, J.-C. (2000). Multimedia content analysis-using both audio and visual clues. *IEEE Signal Processing Magazine*, 17(6):12 –36.

[Wirth, 2000] Wirth, R. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pages 29–39.

[Xiong et al., 1995] Xiong, W., mong Lee, J. C., and gang Shen, D. (1995). Net comparison: An adaptive and effective method for scene change detection.

[Xu et al., 2001] Xu, P., Xie, L., Chang, S.-F., Divakaran, A., Vetro, A., and Sun, H. (2001). Algorithms and system for segmentation and structure analysis in soccer video. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2001)*, pages 721 –724.

[YouTube, 2012] YouTube (2012). Statistics.

[Yuan et al., 2007] Yuan, J., Wang, H., Xiao, L., Zheng, W., Li, J., Lin, F., and Zhang, B. (2007). A formal study of shot boundary detection. In *IEEE Transaction on Circuit and Systems For Video Technology*, pages 168–186.

[Zabih et al., 1995] Zabih, R., Miller, J., and Mai, K. (1995). A feature-based algorithm for detecting and classifying scene breaks. In *Proceedings of the 3rd ACM International Conference on Multimedia*, MULTIMEDIA '95, pages 189–200, New York, NY, USA. ACM.

[Zattoo, 2012] Zattoo (2012). Zattoo erreichte während der Europameisterschaft in Deutschland mehr als 1 Million Nutzer. zuletzt abgerufen am 05.01.2013.

[Zeadally et al., 2011] Zeadally, S., Moustafa, H., and Siddiqui, F. (2011). Internet protocol television (IPTV): Architecture, trends, and challenges. *IEEE Systems Journal*, 5(4):518 –527.

[Zhang et al., 1993] Zhang, H., Kankanhalli, A., and Smoliar, S. W. (1993). Automatic partitioning of full-motion video. *Multimedia Systems*, 1(1):10–28.

[Zhang et al., 1995a] Zhang, H., Low, C., and Smoliar, S. W. (1995a). Video parsing and browsing using compressed data. *Multimedia Tools and Applications*, 1:89–111.

[Zhang et al., 1995b] Zhang, H., Tan, S. Y., Smoliar, S. W., and Yihong, G. (1995b). Automatic parsing and indexing of news video. *Multimedia Systems*, 2:256–266. 10.1007/BF01225243.

[Zhang et al., 1995c] Zhang, H. J., Low, C. Y., Smoliar, S. W., and Wu, J. H. (1995c). Video parsing, retrieval and browsing: an integrated and content-based solution. In *Proceedings of the 3rd ACM international Conference on Multimedia*, MULTIMEDIA '95, pages 15–24, New York, NY, USA. ACM.

[Zhang et al., 1996] Zhang, T., Ramakrishnan, R., and Livny, M. (1996). Birch: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 103–114, New York, NY, USA. ACM.

[Zhong et al., 2004] Zhong, H., Shi, J., and Visontai, M. (2004). Detecting unusual activity in video. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, volume 2, pages II–819 – II–826 Vol.2.

A. The "news"-dataset

This table shows an example for the labels for one "Tagesschau" news show, recorded on September 11th, 2012. The frame numbers indicate the first frame of each shot. The transitions distinguish between cuts (C) and gradual transitions (GT). If the shot does belong to something else than the "news report" class, the class is explicitly named in the last column. "AS" hereby is a short form for "anchorshot".

frame	**transition**	**label**
35	GT	Intro
257	GT	Intro
258	C	AS
1209	C	
1410	C	
1506	C	
1590	C	
1707	C	
1782	C	
1857	C	
1920	C	
2153	C	
2310	C	
2392	C	
2637	C	
2851	C	
3010	C	
3200	C	
3358	C	
3472	C	
3940	C	AS

4402	C	Diagrams
4757	GT	
4928	C	
5015	C	
5144	C	
5395	C	
5527	C	
5638	C	
5899	C	
6013	C	
6096	C	
6155	C	
6565	C	
6620	C	
6680	C	
6766	C	
7121	C	
7256	C	
7369	C	AS
8066	C	
8238	C	
8294	C	
8363	C	
8423	C	
8556	C	
8642	C	
8814	C	
8990	C	
9222	C	
9297	C	
9366	C	
9439	C	
9514	C	
9592	C	
9688	C	
9945	C	

10007	C	
10046	C	
10500	C	AS
11655	C	
11717	C	
11795	C	
11878	C	
11965	C	
12140	C	
12377	C	
12588	C	
12677	C	
12758	C	
12928	C	
12984	C	
13060	C	
13196	C	
13360	C	
13708	C	
13885	C	
13950	C	
14007	C	AS
14337	C	
14607	C	
14657	C	
14810	C	AS
15283	C	
15381	GT	
15495	GT	
15632	GT	
15740	GT	
15876	GT	
15953	C	
16091	C	
16209	C	
16304	C	

16421	C	
16457	C	AS
17299	C	
17435	C	
17491	C	
17557	C	
17622	C	
17693	C	
17769	GT	
17898	C	
17964	C	
18011	C	
18180	C	
18237	C	
18313	C	
18387	C	
18445	C	
18484	C	
18523	C	
18610	C	
19001	C	
19068	C	
19154	C	
19288	C	
19469	GT	
19555	C	AS
19782	C	
20069	C	
20125	C	
20239	C	
20351	C	
20424	C	
20430	C	AS
20628	C	
20691	C	
20810	C	

20864	C	
20916	C	
20986	C	
21048	C	
21117	C	
21174	C	
21227	C	
21313	C	AS
21458	C	Weather
23020	C	AS
23323	C	Extro

B. The "coffee"-dataset

This table shows the labels for one of the coffee capsule videos. Each row contains one event, given by the frame number the event starts with (=number of the frame, on which the capsule can first be seen) plus the correct color label of the capsule.

frame	**label**
99	red
178	yellow
242	blue
294	green
350	black
416	purple
482	red
534	yellow
581	blue
639	green
696	black
757	purple
807	red
875	yellow
923	blue
981	green
1041	black
1098	purple
1147	red
1199	yellow
1255	blue
1304	green
1364	black

1416	purple
1476	red
1527	yellow
1577	blue
1632	green
1685	black
1745	purple
1801	red
1854	yellow
1900	blue
1951	green
2006	black
2060	purple

Figure .1.: Coffee capsules sorted in the order they were placed on the slide.

C. Features extracted with RapidMiner IMMI

frame	: color	: peakcolor	
frame	: color	: peakcolor	: relativecount
frame	: color	: eginess	
frame	: color	: centerofmass	: normalizedx
frame	: color	: centerofmass	: normalizedy
frame	: grayscale	: areafraction	
frame	: grayscale	: kurtosis	
frame	: grayscale	: max	
frame	: grayscale	: mean	
frame	: grayscale	: median	
frame	: grayscale	: minimum	
frame	: grayscale	: skewness	
frame	: grayscale	: standarddeviation	
frame	: grayscale	: histogram	: bin0
frame	: grayscale	: histogram	: bin1
frame	: grayscale	: histogram	: bin2
...	: ...	: ...	: ...
frame	: grayscale	: histogram	: bin9
frame	: (blue , red , green)	: areafraction	
frame	: (blue , red , green)	: eginess	
frame	: (blue , red , green)	: kurtosis	
frame	: (blue , red , green)	: max	
frame	: (blue , red , green)	: mean	
frame	: (blue , red , green)	: median	
frame	: (blue , red , green)	: min	

frame	: (blue , red , green)	: centerofmass	: normalizedx
frame	: (blue , red , green)	: centerofmass	: normalizedy
frame	: (blue , red , green)	: skewness	
frame	: (blue , red , green)	: standarddeviation	
2-frame	: difference_image	: areafraction	
2-frame	: difference_image	: eginess	
2-frame	: difference_image	: kurtosis	
2-frame	: difference_image	: max	
2-frame	: difference_image	: mean	
2-frame	: difference_image	: median	
2-frame	: difference_image	: min	
2-frame	: difference_image	: centerofmass	: normalizedx
2-frame	: difference_image	: centerofmass	: normalizedy
2-frame	: difference_image	: peakcolor	: relativecount
2-frame	: difference_image	: peakcolor	
2-frame	: difference_image	: skewness	
2-frame	: difference_image	: standarddeviation	
frame	: black_and_white	: areafraction	
frame	: black_and_white	: mean	
frame	: black_and_white	: median	
frame	: black_and_white	: centerofmass	: normalizedx
frame	: black_and_white	: centerofmass	: normalizedy
frame	: borders	: areafraction	
frame	: borders	: mean	
frame	: borders	: median	
frame	: borders	: centerofmass	: normalizedx
frame	: borders	: centerofmass	: normalizedy

D. Implemented processors in the streams framework

Processors in Package `stream.image`

This package provides several processors for transforming images (video frames).

Processor `BorderDetection`

This processor performs a border detection on the input image. A pixel is said to be a border pixel, if its' color differs from the color of at least one of its' four neighboring pixels. By increasing the tolerance, the amount of neighboring pixels that have to have the same color value can be decreased.

Parameter	Type	Description	Required
`output`	String	The name/key of the output image is stored. If this name equals the name of the input image, the input image is going to be overwritten.	false
`tolerance`	int	The number of neighboring pixels that may have a different color value, without causing, that the actual pixel becomes recognized as a border pixel. The higher the tolerance is, the less border pixels will be found.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .4.: Parameters of class `BorderDetection`.

Processor `ColorDiscretization`

The processor discretizes the color space of the input image by discretizing each single RGB color channel.

Parameter	Type	Description	Required
`bins`	Integer	Set the number of discrete color values, each channel in divided into.	false
`output`	String	The name/key under which the output image is stored. If this name equals the name of the input image, the input image is going to be overwritten.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .5.: Parameters of class `ColorDiscretization`.

Processor `ColorToGrayscale`

This processor converts a color image to a grayscale image.

Parameter	Type	Description	Required
`output`	String	The name/key under which the output image is stored. If this name equals the name of the input image, the input image is going to be overwritten.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .6.: Parameters of class `ColorToGrayscale`.

Processor `Crop`

This processor crops an image to a new size by cutting away parts of the image.

Parameter	Type	Description	Required
`output`	String	Key/name of the attribute into which the output cropped image is placed, default is 'frame:cropped'.	false
`width`	int	Width of the rectangle to crop, default is 10.	false
`height`	int	Height of the rectangle to crop, default is 10.	false
`x`	int	x coordinate of the lower-left corder of the rectangle for cropping, defaults to 0.	false
`y`	int	y coordinate of the lower-left corder of the rectangle for cropping, defaults to 0.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .7.: Parameters of class `Crop`.

Processor `DetectBrokenImage`

This processor detects broken images and trows them away.

Parameter	Type	Description	Required
`threshold`	double	The fraction of pixels that need to have the 'broken' color to mark this image as broken.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .8.: Parameters of class `DetectBrokenImage`.

Processor `DiffImage`

This processor computes the difference image of the actual image and the image before.

Parameter	Type	Description	Required
`output`	String	The name/key under which the output image is stored. If this name equals the name of the input image, the input image is going to be overwritten.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .9.: Parameters of class `DiffImage`.

Processors in Package `stream.image.features`

This package provides several processors for extracting features from images (video frames).

Processor `AverageRGB`

This processor extracts RGB colors from a given image and computes the average RGB values over all pixels of that image.

Parameter	Type	Description	Required
`includeRatios`	boolean	Sets, if the processor includes the ration between the color channels, or just the average RGB color values. Ratios are (red/blue), (red/green), (green/blue).	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .10.: Parameters of class `AverageRGB`.

Processor `CenterOfMass`

This processor calculates the Center of Mass of one color channel of the image. You can either ask for the absolute x- and y-coordinates of the Center of Mass or the normalized Center of Mass (= absolute Center of Mass / the size of the image).

Parameter	Type	Description	Required
`normalized`	Boolean	Sets, if the processor computes the normalized Center of Mass or the absolute Center of Mass	false
`colorchannel`	String	Sets the color channel, on which the Center of Mass computation is based on.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .11.: Parameters of class `CenterOfMass`.

Processor `ColorHistogram`

This processor computes a color histogram on one color channel of an image.

Parameter	Type	Description	Required
`bins`	Integer	Sets the number of bins the color channel is discretized into.	false
`colorchannel`	String	Sets the color channel the histogram is computed for.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .12.: Parameters of class `ColorHistogram`.

Processor `MedianRGB`

This processor extracts RGB colors from a given image and computes the median value for each color channel.

Parameter	Type	Description	Required
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .13.: Parameters of class `MedianRGB`.

Processor `StandardDeviationRGB`

This processor computes the standard deviation for all three RGB channels. Requires the Average (=Mean) value for all RGB channels to be included already. This can for example be done by using the AverageRGB processor.

Parameter	Type	Description	Required
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .14.: Parameters of class `StandardDeviationRGB`.

Processors in Package `stream.image.filters`

This package provides several processors for applying filters to images.

Processor `SetTransparent`

This processor sets black pixels transparent by setting the alpha value of black pixels to 0. Transparent pixels are not taken into account by some other processors.

Parameter	Type	Description	Required
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .15.: Parameters of class `SetTransparent`.

Processor `Smoothing`

This processor is a filter, smoothing the image. Hence the processor reduces the noise in an input image. Each pixel gets replaced by the average of pixels in a square window surrounding the pixel.

Parameter	Type	Description	Required
`output`	String	The name/key of the output image is stored. If this name equals the name of the input image, the input image is going to be overwritten.	false
`windowSize`	Integer	Sets the window size. The window size determines the neighboring pixels for each pixel, that are averaged. A windowSize of 3 means that 3 times 3 = 9 pixels are taken into account.	false
`weighted`	Boolean	If a weighted smoothing technique is selected, neighbors closer to the pixel to be smoothend are counted with a higher weight.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .16.: Parameters of class `Smoothing`.

Processors in Package `stream.coffee.helper`

This package provides several processors useful for processing the coffee dataset.

Processor `AddCoffeeLabels`

This processor adds the true label to the video frames of one coffee capsule video file. One label is attached to each data item: @label:event, telling, the color of the coffee capsule slipping down the slide or "no event", in case no capsule is shown within the frame (multinominal).
The labels have to be stored in a file, that gets read in before the stream process starts (during the init of this processor).

Parameter	Type	Description	Required
`file`	String	Sets the file the labels are stored in.	false

Table .17.: Parameters of class `AddCoffeeLabels`.

Processor `DatasetGenerator`

This processor creates a dataset, containing only one data item for each coffee capsule event. If the last data item belonged to an event, but the current does not, the minimal RGB values for that event plus the label are returned. All other data items are simply dropped. The minimal color values observed during the event have to be included in the data item already. This can be done using the
stream.coffee.tagging.MaxRGBOverEvent processor.

Processors in Package `stream.coffee.eventdetection`

This package provides processors to detect events and evaluate event detection classifiers.

Processor `EventDetectionEvaluation`

This processor evaluates the quality of a learner, which classified frames as "events" or "no events". As an event is said to be recognized, as soon as at least one frame of the event was classified as "event", the evaluation processors, included in the core streams module, are not sufficient to perform the evaluation.

Parameter	Type	Description	Required
`label`	String	The key, under which the t label is stored.	false
`prediction`	String	The key, under which the predicted label is stored.	false

Table .18.: Parameters of class `EventDetectionEvaluation`.

Processor `ThresholdEventDetection`

This processor classifies the incoming frames as "events" or "no events", based on the value of one color channel. If the average color value of the color channel falls below a given threshold t, the frame is labeled as an "event" frame.

Parameter	Type	Description	Required
`attribute`	String	Tells the processor, on which attribute to base the event detection on.	false
`t`	Integer	Sets the threshold t to a new value.	false
`predictionkey`	String	Sets the key under which the classifier stores the predicted label.	false
`standardvalue`	Integer	Sets the value the attribute has in random frames.	false

Table .19.: Parameters of class `ThresholdEventDetection`.

Processors in Package `stream.coffee.tagging`

This package provides processors useful for processing the coffee dataset.

Processor `MaxRBGOverEvent`

This processor determines the minimal RGB color values observed during the actual event. If the frame does not belong to an event (@prediction:event = false), no action is performed. As soon as a series of event frames start, the minimal color value for all color channels is remembered and the minimal values observed so far are added to the data item. Hence the last data item of the series holds the minimal color values for the whole event.

Processors in Package `stream.news.helper`

This package provides several processors useful for processing the news dataset.

Processor `AddNewsshowLabels`

This processor adds the true label to the video frames of one news show. Three labels are attached to each data item: @label:shotboundary, telling, whether the frame is the first frame after a shot boundary or not (binominal), @label:anchorshot, telling, whether the frame belongs to a shot boundary or not (binominal), and @label:shottype, telling, what concrete type of shot the frame belongs to (multinominal).
The labels have to be stored in a file, that gets read in before the stream process starts (during the init of this processor).

Parameter	Type	Description	Required
`file`	String	Sets the file the labels are stored in.	false

Table .20.: Parameters of class `AddNewsshowLabels`.

Processor `ErrorOutput`

This processor compares a predicted label with the true label. If both labels match, no action is performed. If the prediction deviates from the true label, the id of the frame gets stored. This processor hence helps to get an overview of the data items, which were not labeled correctly. This turned out to be useful to improve the classifiers.

Parameter	Type	Description	Required
`label`	String	Sets the key under which the true label is stored.	false
`prediction`	String	Sets the key under which the predicted label was stored by the classifier.	false

Table .21.: Parameters of class `ErrorOutput`.

Processors in Package `stream.news.sbdetection`

This package provides processors to detect shot boundaries.

Processor `GrayThreshold`

This processor classifies the incoming frames as "shot boundaries" or "no shot boundary" based on the DiffImages of two successive frames. It can only be applied, after the DiffImage has already been calculated. If the average gray value of the DiffImages exceeds a given threshold t, the frame is labeled as a "shot boundary".

Parameter	Type	Description	Required
`t`	Integer	Sets the threshold t.	false
`predictionkey`	String	The key, under which the classifier shall store the predicted label.	false
`graykey`	String	Tells the GrayThreshold processor, where to find the gray value of the pixels.	false

Table .22.: Parameters of class `GrayThreshold`.

Processors in Package `stream.news.anchorshotdetection`

This package provides several processors useful for detecting anchorshots in news videos and creating anchorshot models.

Processor `AnchorshotModelCreator`

This processor learns an image model, which represents anchorshots, by looking for areas in anchorshot frames that have constant color values throughout all incoming anchorshot frames.

Parameter	Type	Description	Required
`model`	String	URL of a file, where the model can be stored on the disk.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .23.: Parameters of class `AnchorshotModelCreator`.

Processor `ApplyDecisionTreeModel`

This processor classifies a news video into anchorshots and news report shots, by applying a decision tree to each example. The decision tree has to be learned using RapidMiner, as not Decision Tree Learner is implemented in the streams framework yet.

Parameter	Type	Description	Required
`predictionkey`	String	Sets the key under which the classifier shall store the predicted label.	false

Table .24.: Parameters of class `ApplyDecisionTreeModel`.

Processor `ApplyImageModel`

This processor classifies a news video into anchorshots and news report shots, by matching each shot against an image, representing an anchorshot model. Pixels, that are black in the model, get ignored, all other pixels get compared pixel-wise. If the difference between the current image and the model does not exceed a given threshold t, the shot is predicted to be an anchorshot.

Parameter	Type	Description	Required
`model`	String	Tells the processor, where to find the model to be matched.	false
`t`	Integer	Sets the threshold t to a new value.	false
`predictionkey`	String	Sets the key under which the classifier shall store the predicted label.	false
`image`	String	The name of the attribute that contains the byte array data of the image.	true

Table .25.: Parameters of class `ApplyImageModel`.

Erklärung

Hiermit erkläre ich, Matthias Schulte, die vorliegende Diplomarbeit mit dem Titel *Real-time feature extraction from video stream data for stream segmentation and tagging* selbständig verfasst und keine anderen als die hier angegebenen Hilfsmittel verwendet, sowie Zitate kenntlich gemacht zu haben.

Dortmund, January 22, 2013