UNIVERSITÄT BONN

Temporal Segmentation of Human Actions in Videos

Dissertation

zur

Erlangung des Doktorgrades (*Dr. rer. nat.*)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich–Wilhelms–Universität Bonn

vorgelegt von

Alexander Richard

aus

Schmallenberg, Deutschland

Bonn 2019

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich–Wilhelms–Universität Bonn

1. Gutachter: Advisor: Juergen Gall
2. Gutachter: Ivan Laptev
Tag der Promotion: 04.10.2019
Erscheinungsjahr: 2019

Abstract

by Alexander Richard

for the degree of

Doctor rerum naturalium

Understanding human actions in videos is of great interest in various scenarios ranging from surveillance over quality control in production processes to content-based video search. Algorithms for automatic temporal action segmentation need to overcome severe difficulties in order to be reliable and provide sufficiently good quality. Not only can human actions occur in different scenes and surroundings, the definition on an action itself is also inherently fuzzy, leading to a significant amount of inter-class variations. Moreover, besides finding the correct action label for a pre-defined temporal segment in a video, localizing an action in the first place is anything but trivial. Different actions not only vary in their appearance and duration but also can have long-range temporal dependencies that span over the complete video. Further, getting reliable annotations of large amounts of video data is time consuming and expensive.

The goal of this thesis is to advance current approaches to temporal action segmentation. We therefore propose a generic framework that models the three components of the task explicitly, i.e. long-range temporal dependencies are handled by a context model, variations in segment durations are represented by a length model, and short-term appearance and motion of actions are addressed with a visual model. While the inspiration for the context model mainly comes from word sequence models in natural language processing, the visual model builds upon recent advances in the classification of pre-segmented action clips. Considering that long-range temporal context is crucial, we avoid local segmentation decisions and find the globally optimal temporal segmentation of a video under the explicit models.

Throughout the thesis, we provide explicit formulations and training strategies for the proposed generic action segmentation framework under different supervision conditions. First, we address the task of fully supervised temporal action segmentation, where frame-level annotations are available during training. We show that our approach can outperform early sliding window baselines and recent deep architectures and that explicit length and context modeling leads to substantial improvements.

Considering that full frame-level annotation is expensive to obtain, we then formulate a weakly supervised training algorithm that uses ordered sequences of actions occurring in the video as only supervision. While a first approach reduces the weakly supervised setup to a fully supervised setup by generating a pseudo ground-truth during training, we propose a second approach that avoids this intermediate step and allows to directly optimize a loss based on the weak supervision. Closing the gap between the fully and the weakly supervised setup, we moreover evaluate semi-supervised learning, where video frames are sparsely annotated.

With the motivation that the vast amount of video data on the Internet only comes with meta-tags or content keywords that do not provide any temporal ordering information, we finally propose a method for action segmentation that learns from unordered sets of actions only. All approaches are evaluated on several commonly used benchmark datasets. With the proposed methods, we reach state-of-the-art performance for both, fully and weakly supervised action segmentation.

Keywords: action segmentation, action detection, weakly supervised learning, Viterbi decoding

Acknowledgements

I would like to thank Juergen Gall for his supervision throughout the last years. I am grateful for his support, for his great ideas, and all the effort he spent on supervising me. I also want to thank him for providing a unique work environment, giving his students all the freedom they want and all advice and guidance they need.

Further, I want to thank Ivan Laptev, who kindly agreed to serve as an external reviewer for my thesis. I also want to thank the other members of my thesis committee, Christian Bauckhage and Jan Boerner.

Special thanks go to my colleagues in the Computer Vision Group at University of Bonn, not only for their great work but also for making our lab such a joyful place. Particularly, I am grateful for countless useful discussions and many valuable collaborations with Hilde Kuehne. I also want to thank Umar Iqbal for great experiences and unique memories during our conference trips, Ahsan Iqbal and Yazan Abu Fahra for believing in me as a supervisor during their Master thesis and for continuing their work by joining the group as PhD students, and Martin Garbade for taking care of all hardware issues. It has been an honor to be part of this group.

Further, I would like to thank my former supervisors Simon Wiesler and Hermann Ney, who let me get in touch with research early during my Bachelors and Masters and who have a large share on my decision to start a PhD in the first place. I am also grateful for an amazing time during my internship at FRL Pittsburgh. Many thanks to Yaser Sheikh, Colin Lea, and all other members of the team who sustainably changed the way I approach new challenges.

I am especially grateful for my family and friends, who always were understanding and considerate during deadlines and more stressful periods. Last but not least, I would like to thank my parents for supporting my every decision and for giving me the opportunity to pursue my dreams from my early childhood on.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

An alphabetically sorted list of abbreviations used in the thesis:

ASR	automatic speech recognition
BPTT	backpropagation through time
C3D	convolutional 3D
CNN	convolutional neural network
CTC	connectionist temporal classification
ECTC	extended connectionist temporal classification
GMM	Gaussian mixture model
GRU	gated recurrent unit
HMM	hidden Markov model
HOF	histograms of optical flow
HOG	histograms of oriented gradients
HTK	hidden Markov model toolkit
I3D	inflated 3D convolutional neural network
IDT	improved dense trajectory
IoD	intersection over detection
IoU	intersection over union
LSTM	long short-term memory
mAP	mean average precision
MBH	motion boundary histograms
MLP	multilayer perceptron
NMS	non-maximum suppression
OCDC	ordered constrained discriminative clustering
RNN	recurrent neural network
STIP	space-time interest points
SVM	support vector machine
TCFPN	temporal convolutional feature pyramid network
TCN	temporal convolutional network
VLAD	vectors of locally aggregated descriptors

Frequently Used Symbols

$\mathcal{C}$	the set of action classes
Γ	a right-regular grammar
$\mathbf{x}_1^T$	a sequence of T feature vectors $x_1, \dots, x_T$

$\mathbf{c}_1^N$	a sequence of N class labels $c_1, \ldots, c_N$
$\mathbf{l}_1^N$	a sequence of N segment lengths $\ell_1, \ldots, \ell_N$
$\mathbf{h}_1^N$	a sequence of N non-terminal symbols of a grammar
$\mathbf{s}_1^N$	a sequence of T HMM states
$n(t)$	a function mapping from frame indices to segment indices
t_n	the index of the last frame of segment n

List of Publications

The thesis is based on the following publications:

A. Richard and J. Gall
Temporal Action Detection using a Statistical Language Model
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016 (Spotlight)

A. Richard and J. Gall
A Bag-of-Words Equivalent Recurrent Neural Network for Action Recognition
Computer Vision and Image Understanding (CVIU), 2017

A. Richard, H. Kuehne, J. Gall
Weakly Supervised Action Learning with RNN based Fine-to-coarse Modeling
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017 (Oral)

A. Richard, H. Kuehne, J. Gall
Action Sets: Weakly Supervised Action Segmentation without Ordering Constraints
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018 (Spotlight)

A. Richard, H. Kuehne, A. Iqbal, J. Gall
NeuralNetwork-Viterbi: A Framework for Weakly Supervised Video Learning
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018 (Spotlight)

H. Kuehne*, **A. Richard***, J. Gall
A Hybrid RNN-HMM Approach for Weakly Supervised Action Recognition
IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019
(indicates equal contribution)*

Code and videos of presentations related to these publications can be found online at
`https://alexanderrichard.github.io`.

Chapter 1

Introduction

Contents

1.1 Motivation

Until about fifteen years ago, research in computer vision was mainly focused on image processing. Video data was treated somewhat negligently, partially due to the lack of resources that would allow to process real world video data in reasonable time but also because its impact on everyday life was much smaller than it is today.

With the availability of video platforms such as Vimeo or YouTube and with cameras integrated in most smart phones, however, the situation changed. Today, user generated video content can be shared on Facebook, YouTube, and other platforms by everyone with access to an Internet connection. According to a keynote of YouTube CEO Susan Wojcicki from 2015, more than 400 hours of video data are uploaded to YouTube every minute (VidCon, 2015). In general, video streaming accounts for 22% of upstream traffic on the Internet (Sandvine Inc., 2018). Given this steadily increasing amount of video data, the interest in analyzing it has grown significantly.

Since videos often revolve around humans as protagonists, human actions are of special importance. Particularly user generated data needs to be analyzed before sharing it publicly. For example, videos showing violence, illegal activities, and adult content should be filtered. Given the sheer amount of data, manual filtering of such content is impossible and algorithmic solutions need to be applied. Another practical use case that requires detecting and classifying of human actions in videos arises in surveillance. Suspicious and malicious behavior needs to

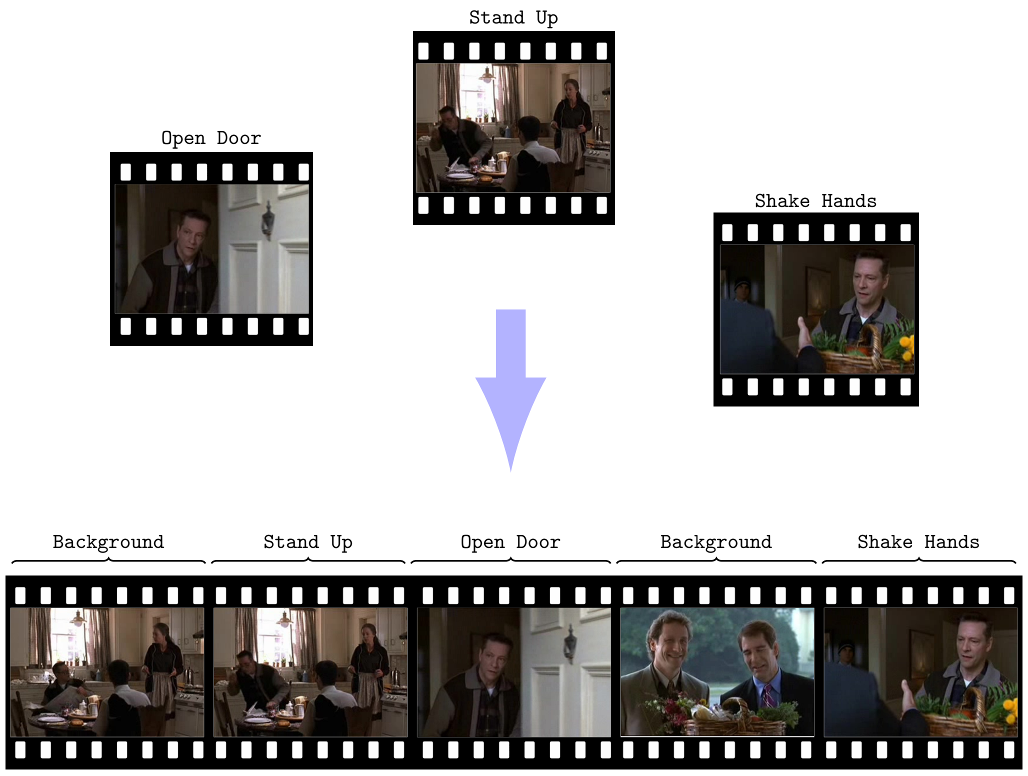


Figure 1.1: Methods for action recognition on pre-segmented clips already work well. Moving towards temporally untrimmed videos with multiple action instances, however, is not trivial as action boundaries are not known in advance and background frames can have similar appearance as action instances.

be detected and categorized ideally in realtime to allow for immediate reaction and to inhibit such actions. Quality control in production processes can also be guided by temporal action detection and recognition. If a process requires the completion of certain steps, algorithms for video analytics can identify errors and trigger reminders or warnings, effectively decreasing the amount of deficient products. Moreover, analyzing human actions in videos is an important element of human-machine interaction. The ability to recognize and anticipate actions and human behavior is crucial for autonomous agents to plan their own actions. Driver assistance systems, for instance, can leverage the information if the driver is yawning or distracted by using his phone and trigger a warning. In autonomous driving, analyzing video streams can help to anticipate dangerous situations like a child playing close to the street. Finding actions in videos can also be used for content-based search. Video queries can be based on the actions occurring within the video rather than on keywords in the title or meta-tags. Similarly, recognizing actions in videos can also help to enhance automatic highlight extraction and video summarization.

Considering the complexity of temporal action segmentation in videos, early works mainly focused on a simplified problem and addressed classification of human actions in pre-segmented clips (Schuldt et al., 2004; Wang and Schmid, 2013). While recent advances using deep neural networks (Simonyan and Zisserman, 2014; Wang et al., 2016; Carreira and Zisserman, 2017) and the availability of large-scale datasets (Kay et al., 2017) have led to outstanding improvements and current systems already reach accuracies far above 90%, classification of pre-segmented clips is an unrealistic scenario for most applications. Real-world videos are temporally untrimmed and contain a variety of different action instances as well as background frames, see Figure 1.1. Therefore, the interest in finding temporal segmentations of such untrimmed videos greatly increased over the last years. First approaches of applying the well studied methods for pre-segmented action recognition in combination with sliding windows (Rohrbach et al., 2012; Oneata et al., 2014) quickly reached their limits. Currently, temporal models that are purely data-driven and allow for end-to-end learning are the dominating approach (Yeung et al., 2016; Lea et al., 2017; Chao et al., 2018).

One of the major challenges nowadays is to analyze growing amounts of video data. A good approach for analyzing human actions needs to have certain properties. First, there is a hierarchical structure and temporal long-range dependencies underlying human actions the approach should be aware of. Second, given the vast amount of available video data, algorithms need to be able to learn without or only with very little human guidance. We discuss these challenges in the next section.

1.2 Challenges

Recognizing human actions in videos is a hard problem. On the one hand, actions can include complex interactions with objects or other humans. On the other hand, the same action can be performed in various environments and under different conditions, leading to a high intra-class variation. Furthermore, actions are inherently fuzzy in various ways. First, there are smooth transitions between an action belonging to a class or not. For instance, consider the action *drinking*. This may refer to a typical scenario of drinking from a cup or a bottle but it can also refer to drinking from a river or lake. Second, temporal transitions between actions are not well defined and humans usually disagree about the exact start and end time of an action. While this can be an issue once action segmentation systems get extremely accurate, at the current state it is found not to be critical (Alwassel et al., 2018). Finding actions in a temporally untrimmed video also poses a combinatorical problem as the number of possible action sequences and boundary positions grows exponentially in the number of action classes and the video duration.

Moving towards temporal action segmentation systems that are good enough for a wide variety of practical applications requires to address two major issues. First, since humans perceive actions in a hierarchical way, action segmentation systems need to model different temporal granularities and allow for long-range dependencies between low-level actions. Second, an increasing amount of video data that needs to be processed also leads to an increasing number of relevant action classes and more intra- and inter-class variations. Therefore, a large amount of training data is required to learn data-driven models. Manually annotating such a vast amount of data, however, is expensive and frequently infeasible. Action segmentation algorithms therefore need to be able to learn with very little supervision only. We discuss the two issues in more detail in the following.

1.2.1 The Hierarchical Nature of Actions

A major problem in the field of action recognition is the definition of action classes and the long-range dependencies between actions. An action does not only depend on a subjects body motion or pure human-human or human-object interaction but also on other possibly non-visual context in the scene. Early philosophical and psychological works on human actions already identify actions to have a hierarchical structure (Barker and Wright, 1954) and that there are basic and non-basic acts but without knowledge of the circumstances or context, one can only recognize the basic acts (Danto and Morgenbesser, 1963; Danto, 1965). Vallacher and Wegner (1987) find that actions are perceived on different levels and that the perceived level of an action depends on its context and difficulty. In general, humans tend to prefer high-level actions. For instance, a sequence of *tipping a finger on a phone* is rather identified as the action *dial the phone* or even *call person X* if such high-level identities can be deduced from the context. Difficult or unfamiliar actions, on the contrary, are more likely to be perceived as multiple low-level identities.

Both, the definition of action classes and modeling high-level temporal dependencies, are therefore a crucial issue in action recognition and segmentation. Most datasets for action recognition on pre-segmented clips define a set of high-level action instances such as performing a sport activity or playing an instrument (Soomro et al., 2012), or mid-level activities such as *hug*, *clap*, or *eat* (Kuehne et al., 2011). Datasets with a focus on temporal action segmentation oftentimes feature an implicit or explicit hierarchy. The Breakfast dataset of Kuehne et al. (2014), for instance, has a video-level action describing a breakfast dish, mid-level descriptions such as *pour milk*, and low-level actions such as *open cap*. The 50 Salads dataset of Stein and McKenna (2013) follows a similar approach and decomposes the activity of making salad into mid- and low-level actions. In such datasets, there are explicit temporal dependencies between low-level and mid-level actions that typically characterize a high-level action, such as the sequence of pouring milk into a bowl, adding cereals, and stirring them characterizes the high-level activity of preparing cereals. The recently published AVA dataset (Gu et al., 2018) follows another principle and mainly relies on low-level actions to reduce the influence of the context. Still, such low-level actions have implicit temporal and hierarchical dependencies. For instance, *pick up* followed by *hold* and *put down* relates to a more high-level human-object interaction. Either way, such dependencies need to be modeled.

1.2.2 Supervision

In the era of deep learning, availability of training data is crucial. The current state-of-the-art in action recognition relies on a deep neural network that is trained on $240,000$ video clips (Carreira and Zisserman, 2017) and covers 400 different action classes. Ideally, datasets for temporal action segmentation should also be large-scale, comprising a few hundred thou-

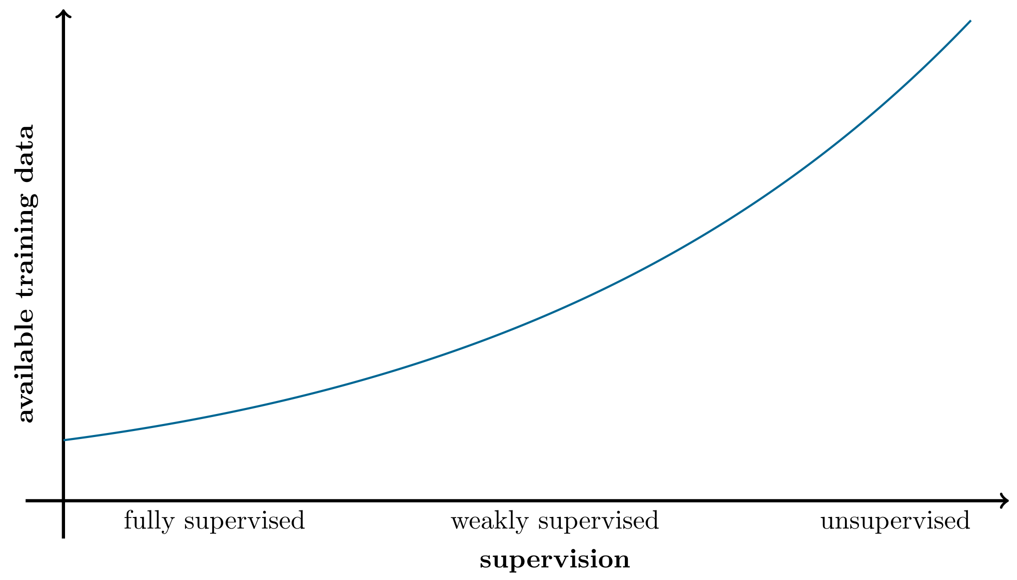


Figure 1.2: The relation between supervision and availability of training data. Fully supervised approaches require precise frame-level annotations that are expensive to obtain and therefore only a limited amount of training data can be provided. Less supervision means lower annotation cost per video, increasing the actual amount of data that can be provided for training.

sand videos from different domains and with different appearance as well as a large amount of action classes. A system for analyzing YouTube videos, for instance, must not only be able to recognize actions in professional television broadcasts but also in amateur videos. Moreover, only a few applications such as surveillance videos have static cameras or a small amount of relevant classes. Mostly, videos have to be analyzed in changing locations with different lighting conditions and varying appearance, with moving cameras, and with a large amount of possible action classes. Training data must capture all these different aspects and therefore, collecting large-scale datasets is inevitable.

Most current approaches for temporal action detection and segmentation require full supervision, i.e. a per-frame labeling or an exact labeling of the action boundaries. Gathering large-scale training data with a frame-level annotation of actions, however, is extremely costly. Widely used benchmark datasets therefore are typically restricted in their domain and size. The Breakfast dataset (Kuehne et al., 2014), 50 Salads (Stein and McKenna, 2013), and MPII Cooking (Rohrbach et al., 2012), for instance, are restricted to kitchen environments and only contain between few dozen and a few thousand videos. In general, the more supervision is required, the less training data usually is available, see Figure 1.2. Therefore, the key to enabling analysis of human actions on unconstrained large-scale data is the development of algorithms that do not require full supervision but can learn from weak annotations or, ideally, from unlabeled data in an unsupervised fashion.

Reducing the amount – and therefore also the cost – of annotation can be achieved in various ways. For example, Gu et al. (2018) label only a single keyframe per second. While this is effective for medium sized datasets, it still does not scale to arbitrarily large datasets. A more scalable approach is to not provide any exact temporal annotations at all but only an ordered sequence of actions that occur in the video. For some video sources such as movies or documentaries, this information can be mined automatically (Laptev et al., 2008; Duchenne et al., 2009). With the availability of high end speech recognition software, similar annotations can also be extracted for arbitrary clips as long as there are semantic correspondences in the visual and acoustic signal. In this context, the Epic Kitchen dataset (Damen et al., 2018) is worth mentioning. The authors asked participants to record themselves in the kitchen and to record a narrated description of their actions afterwards and therefore have semantic audio-video correspondences in their dataset. These narrations have then been used to guide the annotation of actions in the videos. Unfortunately, there is no segmentation task defined for the dataset because not all action instances have been labeled.

Alternatively to subtitles or audio signals as a source of weak supervision, meta-tags can be exploited to define video-level action labels. Since most clips on YouTube or Facebook are equipped with meta-tags, they provide a large source for high-level action annotations. Scalable action segmentation methods have to be able to learn from such weakly annotated data only.

1.3 Contributions

In this thesis, we address both problems, i.e. modeling temporal long-range dependencies and hierarchical relations between actions as well as the development of weakly supervised methods that pave the way for learning from large amounts of video data. The main contributions of the thesis are outlined in the following.

1.3.1 A Generic Framework for Temporal Action Segmentation

Most current approaches to temporal action segmentation are based on end-to-end trainable neural networks and use either temporal segment proposals (Zhao et al., 2017) or temporal convolutional networks with a limited receptive field for segment detection (Lea et al., 2017). These approaches all have in common that they rely on local, possibly suboptimal decisions for action segments.

Inspired by common practice in automatic speech recognition (ASR), where speech is modeled using an acoustic model and a language model (see e.g. Trentin and Gori (2001)), we propose a generic framework that allows for temporal action segmentation based on a factorization into a visual model, a length model, and context model. The three models can be seen as a hierarchical decomposition of the problem. Low-level temporal dependencies within an action are modeled by the visual model. The length model guides the segmentation algorithm on a mid-level by ensuring reasonable durations of actions. Long-range dependencies are finally addressed by the context model that combines low-level action instances over the complete video and thereby captures high-level relations. The framework allows to integrate a wide variety of recent action recognition models such as convolutional neural networks

(CNNs) or recurrent neural networks (RNNs). At the same time, it allows for an efficient and effective decoding that finds the globally optimal segmentation given the underlying models rather than relying on local suboptimal decisions. We apply the framework for different action segmentation related tasks and with different levels of supervision.

1.3.2 Advancing Fully Supervised Action Segmentation

In a next step, we address the task of fully supervised action segmentation using the generic framework. Therefore, we evaluate different kinds of length models and propose to model the context of actions by m-grams, which are widely used in language modeling. Deep neural networks that have pushed classical action recognition to new limits recently can easily be integrated into the framework as visual models. Based on a modification of the Viterbi algorithm (Viterbi, 1967), we derive an efficient algorithm that finds the optimal segmentation. We evaluate our method with traditional hand-crafted features proposed by Wang and Schmid (2013) and also provide results with deep features extracted from an I3D CNN (Carreira and Zisserman, 2017). We further enhance our approach by a bag-of-words equivalent neural network as segment classifier. For the latter, we achieve state-of-the-art performance on Thumos, a challenging benchmark for fully supervised action detection.

1.3.3 Weakly Supervised Action Segmentation using Action Transcripts

As mentioned before, effectively learning from a huge amount of video data requires systems that do not demand the availability of framewise labeled training data. We therefore address the problem of weakly supervised action segmentation, where only action transcripts are provided during training. Thus, in contrast to fully supervised approaches, no frame-level annotation is used and the temporal order in which action classes occur is all that is known for the training videos. While early methods approach weakly supervised action segmentation with discriminative clustering (Bojanowski et al., 2014) or traditional ASR systems (Kuehne et al., 2017), we rely on the proposed generic framework. In a first step, we propose a fine-to-coarse model that uses a recurrent neural network to model local changes in the video and a coarser model that combines subactions to actions. The lack of frame-level annotation during training is addressed by an iterative bootstrapping approach that alters an inference and training step. In a second step, we simplify the initial model and propose a training algorithm that resembles the typical steps of neural network training, i.e. forwarding, loss computation, and backpropagation. Using the advanced model, related weak learning approaches are outperformed by a huge margin.

1.3.4 Weakly Supervised Action Segmentation using Action Sets

Action transcripts require either manual annotation or some temporal information such as subtitles to mine them. Most large video collections on the Internet, however, are only equipped with meta-tags. These tags can be seen as an unordered set of actions or events that occur in the video. Thus, aiming to further reduce the amount of supervision, we remove the temporal ordering constraints. Instead of an ordered sequence of actions, now only the set of action classes occurring in video is provided during training. While this may seem like a minor change in the level of supervision, it actually increases the space of possible solutions exponentially since every permutation of actions from the given set is possible. Moreover, without ordering constraints and frame-level annotation, the length model and context model need to be estimated heuristically. We propose a simple yet effective method to train the generic framework using this kind of weak supervision.

1.4 Thesis Structure

The remainder of the thesis is structured as follows.

In **Chapter 2**, we discuss related work, starting from classical action recognition on pre-segmented clips and advancing to recent approaches for temporal action segmentation. The chapter also contains a detailed description of the datasets used in this thesis.

In **Chapter 3**, the problem of temporal action segmentation is formally defined and preliminary work is discussed. Particularly, we introduce evaluation metrics used on the various datasets and outline their differences. We also describe the most relevant feature extraction methods that find application in our methods, give a brief introduction into recurrent neural networks, and show that a commonly used traditional feature quantization method is essentially a special case of a recurrent neural network. The latter is based on the publication (Richard and Gall, 2017).

Chapter 4 contains the description of the general framework including the decomposition into context model, length model, and visual model. Moreover, a basic version of the Viterbi algorithm including a pseudo-code implementation is derived. The algorithm is refined in later chapters to fit the explicit models.

The task of fully supervised temporal action detection is covered in **Chapter 5**, building upon the previously introduced framework and instantiating explicit length and context models. We evaluate different length models and discuss the interdependencies of the three model components. Chapter 4 and Chapter 5 are based on work published in (Richard and Gall, 2016).

In **Chapter 6**, we propose a first method for weakly supervised action segmentation. It features a fine-to-coarse architecture that is based on recurrent neural networks and a hidden Markov model. In order to further boost the performance, we also add a length regularization that prevents subactions from being unreasonably long. This part of the thesis in mainly based on the publications (Richard et al., 2017; Kuehne et al., 2019).

Simplifying that method, in **Chapter 7**, we replace the hidden Markov model by an explicit length model. We further identify drawbacks in the training procedure and propose a more stable learning approach. The approach does not only reach state-of-the-art performance for weakly supervised action segmentation but also allows for incremental learning. Additionally to the weak learning setup that is based on action transcripts, we also discuss semi-supervised learning where video frames are sparsely annotated. The chapter is based on work published in (Richard et al., 2018b).

In **Chapter 8**, we further reduce the amount of annotation and use unordered action sets as only supervision. Due to the severe lack of annotated data, we propose heuristics to learn the model components and evaluate how text sources can complement the learning from unordered actions. The basis of this chapter is the publication (Richard et al., 2018a).

Chapter 9 concludes the thesis with a discussion of the generic framework and an outlook on future research directions that potentially enable self-supervised learning and anticipation of future actions.

Chapter 2

Related Work

In this chapter, we discuss the most relevant related work for this thesis. Being the foundation for temporal action segmentation, we start with a review of classical action recognition on pre-segmented clips. We then discuss different approaches to temporal action segmentation in both, fully and weakly supervised training setups, and introduce the datasets used throughout the thesis.

Contents

2.1 Classical Action Recognition

In classical action recognition, it is usually assumed that the video clips are already pre-segmented and contain a single action instance only. The task then reduces to a pure classification task. Early works focus on hand-crafted feature extraction and find action classes using nearest neighbor searches or bag-of-words models. More recent deep architectures, on the contrary, solve the problem in a single neural network.

Action recognition of pre-segmented clips is the foundation of current action segmentation approaches that can deal with temporally untrimmed videos. The approaches presented in this thesis also rely on sophisticated frame-based or clip based action classifiers, which we therefore review in more detail in the following.

2.1.1 Early Approaches

While for image recognition, successful feature descriptors such as SIFT (Lowe, 1999) were already available, action recognition requires feature extraction not only in the spatial domain but also in the temporal domain. Early approaches to action recognition therefore focused on either spatio-temporal feature extraction or temporal modeling of framewise features. For the latter, some works refer to the success of hidden Markov models (HMMs) in speech recognition and adapt the idea to video streams, see Weinland et al. (2011) for a survey. A work by Brand et al. (1997), for instance, identifies hand movements as driving components for human actions and proposes to use a coupled HMM in order to model the movement of both hands. They evaluate their approach on two-handed gestures and show that they can recognize basic hand movements. Moving towards full body motion, Chen et al. (2006) propose a star skeleton that is extracted for each frame based on the body shape which is obtained by background subtraction. With an HMM for each action class, the authors then model likelihoods of temporal sequences of star skeleton features.

Avoiding HMMs, some approaches apply template matching where templates for action classes are based on spatio-temporal filters (Rodriguez et al., 2008). Other approaches rely on representing the video by a fixed size feature set and apply a nearest neighbor classifier to recognize action classes. Masoud and Papanikolopoulos (2003), for instance, sample a fixed number of frames from a video and use recursive filtering to extract framewise features. After learning a PCA, test videos and train videos are compared in their respective eigenspaces and the closest match from the training set is used to determine the action class. The approach of İkizler et al. (2008) has a stronger focus on the human body. The authors propose to extract line-based features that represent the contours of the human body and enhance the representation with histograms of optical flow. In order to address the problem that videos are of different temporal length, they score the frame features within a sliding window using a support vector machine (SVM) and select the action detected in the window with the highest response. Efros et al. (2003) state that videos are mostly figure-centric and propose to use tracking and video stabilization before a classification step.

One of the most relevant general feature extractors for videos are space-time interest points (STIP) that have been proposed by Laptev and Lindeberg (2003); Laptev (2005). Inspired by the Harris detector for 2D interest points (Harris and Stephens, 1988), a similar method is proposed to extract interest points in a spatio-temporal 3D volume. These features have first been used for action recognition in combination with a bag-of-words (BoW) approach and SVM classification in Schuldt et al. (2004). This combination of BoW and SVM has led to a huge leap forward in the field of action recognition thereafter. While previous works mainly concentrated on videos that have been recorded under laboratory conditions, i.e. with constant illumination, static background, and single persons only, more realistic videos could be processed now. The first work to use unrestricted movie scenes rather than laboratory scenarios was presented by Laptev and Pérez (2007).

Given the success of bag-of-words models and SVM classifiers, there was a strong focus on the development of spatio-temporal feature extraction algorithms. Building on basic low-level image features, Niebles and Fei-Fei (2007) propose a high-level constellation of parts to approach a more structured representation of a video. In a work by Wang et al. (2013), trajectories of pixels through the spatio-temporal video volume are densely sampled and appearance as well as motion descriptors are extracted along these trajectories. An improved version of this work by Wang and Schmid (2013) adds stabilization to the trajectory extraction and replaces the bag-of-words pipeline with Fisher vectors (Perronnin and Dance, 2007), a more advanced feature quantization approach. Building upon the success of improved dense trajectories, Peng et al. (2014) propose stacked Fisher vectors, a hierarchical model that first encodes large video subvolumes as Fisher vectors and in successive layers encodes the resulting vectors again. Improved dense trajectories have been the de-facto state-of-the-art in action recognition until deep neural networks first showed better performance.

2.1.2 Deep Learning for Action Recognition

Deep neural networks are responsible for a huge leap forward in image classification and object recognition (Krizhevsky et al., 2012). For action recognition, however, the success of deep learning proceeded more slowly as capturing temporal dependencies and motion cues with a convolutional neural network is a non-trivial task. The work of Bilen et al. (2016) therefore aims at representing a video clip as a single dynamic image which encodes motion and temporal cues in the video, such that well studied CNN architectures from image processing can be applied for video classification. In another approach to identify how deep learning can help action recognition, Jain et al. (2015) successfully show that deep CNN features from object classification can lead to improvements on action recognition benchmarks. Identifying the size of action datasets as a limiting factor for deep learning, Karpathy et al. (2014) collect a large scale dataset with one million sports related clips and 487 classes. They show that frame-based CNNs can learn strong features and that transfer learning between the large scale video dataset and smaller action recognition benchmarks is possible.

In an attempt to incorporate temporal information into CNNs, some works focus on 3D convolutions. Early approaches like the version of gated Boltzman machines (Memisevic and Hinton, 2007) for unsupervised features learning proposed by Taylor et al. (2010) try to extract spatio-temporal patterns using 3D convolutions that can be used as features for any classification module. Ji et al. (2013) propose a first 3D convolutional neural network for action recognition but do not outperform hand-crafted features. A succeeding work of Tran et al. (2015) proposed an advanced convolutional 3D (C3D) architecture which generates features that are complementary to improved dense trajectories and lead to better performance when combined.

However, all those deep architectures could not consistently outperform improved dense trajectories. Consequently, efforts have been made to combine the benefits of deep learning with the strengths of hand-crafted features. To this end, Lev et al. (2016) build upon Fisher vectors of improved dense trajectories which have been a major improvement over the original dense trajectories from Wang et al. (2013). Fisher vectors summarize a collection of feature vectors based on the gradient of a Gaussian mixture model (GMM) but do not take temporal orderings of these vectors into account. The authors therefore propose to train a recurrent neural network on the set of feature vectors obtained by improved dense trajectories and summarize the sequence using the gradient of the RNN instead of the gradient of the GMM. Another hybrid approach is proposed by Wang et al. (2015), who find that CNNs extract strong features but the strength of improved dense trajectories are the trajectories of pixels moving through the spatio-temporal volume. Thus, they propose to aggregate features from CNN feature maps along those trajectories rather than hand-crafted descriptors.

With the insight that optical flow is crucial for deep architectures in action recognition, CNNs could finally outperform hand-crafted features. The first successful purely CNN based approach that achieves similar performance as improved dense trajectories is the original two-stream network from Simonyan and Zisserman (2014), who use two parallel CNN streams, one working on RGB frames and the other on a stack of 10 consecutive optical flow fields. Driven by the success of this architecture, a series of modifications and improvements emerged in the subsequent years. While the original two-stream network relies on a simple temporal averaging of the output scores to obtain a video-level class prediction, Ng et al. (2015) extend the architecture by a long-short-term-memory (LSTM) network to aggregate the scores along the temporal axis and show promising results. Based on the success of vectors of locally aggregated descriptors (VLAD, Jégou et al. (2010)), Girdhar et al. (2017) propose a fully differentiable layer that pools features from a two-stream network over space and time. Temporal segment networks (Wang et al., 2016) build upon two-stream networks and process multiple temporal snippets that are randomly chosen from uniformly sampled video segments in order to process longer videos. The scores of each snippet are aggregated according to a consensus function. On standard action recognition benchmarks, temporal segment networks lead to a considerable improvement over the original two-stream architecture.

Other works that extend the two-stream architecture introduce motion gated appearance streams (Feichtenhofer et al., 2017b). Therefore, multiplicative connections from the motion stream to the appearance stream are established. Moreover, in Feichtenhofer et al. (2016), the authors show that the size of the network can be reduced by an earlier fusion of the two streams without loss of performance. In an attempt to model a larger temporal context, Feichtenhofer et al. (2017a) build a two-stream net using the ResNet architecture (He et al., 2016) with temporal residual convolutions and thereby expand the temporal receptive field of the network. The currently best performing method for action recognition is an inflated 3D convolutional neural network (I3D, Carreira and Zisserman (2017)) that combines 3D convolutions with the idea of two streams for motion and appearance. The network is trained on a large-scale dataset with 400 classes and $240,000$ videos and has proven well for transfer learning and extracting generic video features.

In this thesis, we mainly rely on the recent I3D architecture for features on datasets with a strong appearance bias and on Fisher vectors of improved dense trajectories as generic features for weakly supervised approaches.

2.2 Temporal Action Segmentation

Temporal action segmentation is the task of finding and labeling action instances in videos. The task is the natural extension of classical action recognition and generally much harder as it is not a pure classification problem but also a detection problem. There is also work on spatio-temporal action detection (Gkioxari and Malik, 2015; Kalogeiton et al., 2017; Singh et al., 2017), localizing actions not only in the temporal but also in the spatial domain. However, these approaches do not focus on long-range temporal context and dependencies between actions but are mostly restricted to localizing a single or very few action instances in a video. The works on spatio-temporal detection therefore focus more on the spatial detection of actions and how these spatial locations change through time and are of subordinate relevance for the temporal models presented in this thesis.

2.2.1 Early Approaches

Pioneering approaches for temporal action segmentation rely on thresholding of action scores and sliding windows. In an early approach, for instance, Bobick and Davis (1996) propose to compute motion history images to capture the temporal context of motion. They extract Hu moments on those images and apply a nearest neighbor classifier to find the best matching action class. In order to filter out frames that do not contain any action, a threshold on the maximal distance to the nearest neighbor is used. The work has later been extended to multiple views (Davis and Bobick, 1997). Ali and Aggarwal (2001) represent human motion by angles between the torso, upper leg, and lower leg, and classify frames either as a breakpoint or a non-breakpoint. A nearest neighbor classifier is then used to label the segments between two breakpoints.

Given the advances in action recognition on pre-segmented clips, sliding window approaches that have been successful in object detection (Viola and Jones, 2001) were soon adopted to the temporal domain. Ke et al. (2005), for instance, use an action classifier based on volumetric features in combination with a spatio-temporal sliding window. Application of a Gaussian filter over the responses and a thresholding to filter background frames then lead to the final segmentation.

Addressing action detection in realistic videos, where actions can occur in different spatio-temporal regions, Laptev and Pérez (2007) propose keyframe priming as an efficient approach. Training a detector for action keyframes, spatio-temporal volumes of different temporal lengths are evaluated around the keyframes, such that an exhaustive sliding window search is not required.

With the availability of larger datasets specifically designed for temporal action segmentation, the quality of such systems rapidly increased (Rohrbach et al., 2012; Oneata et al., 2014; Wang et al., 2014; Karaman et al., 2014). However, all of these approaches have in common that they rely on sliding windows and non-maximum suppression to remove overlapping and low scoring action segments. Works by Ni et al. (2014) and Lan et al. (2015) started to incorporate hierarchical elements into the segmentation process. The first work, for instance, identifies interaction between hands and objects as a coarse indicator for actions and therefore localizes objects that are being manipulated in the video and then uses this information to refine action localization. In the second work, clustering of spatio-temporal action proposals and discriminative clustering is applied to find mid-level action elements from higher-level videos. These models, however, are still not exploiting the inherent contextual dependencies between actions in videos.

2.2.2 Temporal Action Localization and Detection

Temporal action localization, also referred to as temporal action detection, is an instance of action segmentation and is sometimes used interchangeably. Mostly, the term refers to detecting sparse actions in a video. In contrast to other works on action segmentation, action localization is mainly applied to datasets that have a huge background portion and only a single or very few different action classes per video. The number of action instances, however, can be large. Standard benchmark datasets are Thumos (Idrees et al., 2017) and ActivityNet (Caba Heilbron et al., 2015), which do not capture long-range contextual dependencies between action instances.

While approaches for classical action recognition are typically designed to work on short clips, methods for action localization face the problem of reliably detecting action boundaries and separating background from action in the temporal domain. Therefore, some works combine segments along the temporal axis using recurrent neural networks, see e.g. Yuan et al. (2016). However, Singh et al. (2016) study LSTMs for temporal action localization and find that they typically utilize not more than eight seconds of video, which is too short for many applications.

Therefore, most approaches rely on a less generic yet effective paradigm. Instead of analyzing the complete video at once, shorter video segments are analyzed independently of each other and scored by a neural network. Frequently, post-processing in form of non-maximum suppression (NMS) to remove overlapping segments is necessary. Zhao et al. (2017), for instance, use segment proposals and propose structured segment networks, where the proposal is divided into start, main, and ending stages. An activity classifier then determines the action class and a completeness classifier determines if the segment covers all frames of the action. In a multi-stage approach, Shou et al. (2016) process multiple segments per video that are obtained by a sliding window. A proposal network scores these segments as either action or background. In the next stage, a classification and localization module is then used for temporally locating actions. A similar approach has been proposed by Gao et al. (2017a). They extract features on short video units, create a clip pyramid at different temporal scales, and a network then classifies the actions in the segments and regresses exact temporal boundaries. Gao et al. (2017b) extend the work by feeding clip proposals back into the network for further refinement. For these approaches, the final segmentation is obtained after post-processing via NMS.

A common property of neural networks for video features such as C3D is that they compress not only the spatial but also the temporal domain. Shou et al. (2017) therefore propose an integrated convolution-deconvolution architecture that simultaneously downsamples the spatial domain and upsamples the temporal domain to obtain more precise localizations.

Addressing the problem of finding consistent segments with reasonable start and end points, Yuan et al. (2017) model start, middle, and end frames explicitly and use structured maximal sums to find the best scoring segments in a video.

A major drawback of those methods is that evaluating all possible video segments is expensive and easily inhibits realtime performance. Therefore, Caba Heilbron et al. (2017) propose a cascade that prunes segments that are – based on object and scene context – unlikely to contain an action. This pruning step does not need to process a spatio-temporal volume but can operate on a frame basis and is therefore fast. Another solution to the runtime problem is proposed by Buch et al. (2017), who output segment proposals with a single pass over the video. In their approach, features are extracted on small temporal volumes and an RNN outputs scores for segments with multiple fixed lengths that possibly end at that point.

In object detection, a series of works starting with RCNNs and going towards the popular Faster-RCNN (Girshick et al., 2014; Girshick, 2015; Ren et al., 2015) led to the proposal generation process being integrated into the network that also classifies the objects. Inspired by this development, Xu et al. (2017) propose R-C3D, which is a combination of Faster-RCNN and C3D. A 3D convolutional neural network extracts features and a proposal subnet predicts for each temporal position a set of anchor segments. If an action is contained in those anchor segments, the classification subnet outputs the start and end times and action class. Sticking even more to the original Faster-RCNN, Chao et al. (2018) transfer the idea into the one-dimensional space of temporal action segmentation with a multi-scale architecture that increases the receptive field and thus allows for huge variations in the action lengths.

Although the algorithms proposed in this thesis primarily address action segmentation where long-range context between action classes is crucial, we compare to these rather local approaches in Chapter 5. On a standard benchmark for action detection, we show state-of-the-art results although the method is substantially different from the above mentioned, mostly proposal-based approaches.

2.2.3 Action Segmentation with Long-range Dependencies

Action segmentation on untrimmed videos with dense action segments requires to take context into account and to model temporal relations of input frames that span a long period of time. The previously discussed approaches that mainly focus on finding action boundaries and locating individual segments quickly reach their limits when long-range context and interacting action segments of different classes characterize the input videos.

Therefore, temporal convolutional networks (TCNs) with large receptive fields recently gained increased attention. Lin et al. (2017b), for example, use input features extracted from a two-stream network and a 3D convolutional network and feed them into a TCN that compresses the features in the temporal domain in each layer. Prediction anchors are then returned at each level of temporal compression, such that short and precise segments can be inferred from early, less downsampled temporal resolutions, as well as long segments based on late layers with stronger compression. Dai et al. (2017) extract two-stream features and generate proposals at different scales by a sliding window approach. These are then forwarded through a TCN to capture temporal patterns and rank the proposals.

Using TCNs in a hierarchically motivated context, Lea et al. (2016) capture mid-range motion with spatio-temporal convolutions that span a receptive field of ten seconds. High-level temporal action segments and transitions between them are modeled by a semi-Markov model. In Lea et al. (2017), two TCN architectures are analyzed that allow to model even high-level dependencies directly in the network. Following the idea of WaveNet (Van Den Oord et al., 2016), an extraordinary large receptive field allows to learn long-range context in the video. This can be achieved by dilated temporal convolutions that keep the video resolution fixed in each layer or by an encoder-decoder architecture that achieves a large receptive field by temporal pooling and upsampling operations. In an empirical evaluation, the encoder-decoder TCN is found to be the stronger model. Combining the benefits of both ideas, Lei and Todorovic (2018) design a network with a residual stream that processes the video at full resolution and a temporal pooling stream that uses deformable convolutions to incorporate context at different temporal scales. Both streams are combined to allow for precise segment boundaries while still benefitting from temporally compressed high-level context information.

The approaches proposed in this thesis also focus on cases where long-range context is important. In contrast to the aforementioned methods, however, we use a different approach that allows for explicit context and length modeling and therefore is not bound to a predefined receptive field as for TCNs but can model dependencies over an arbitrarily long range.

2.2.4 Globally Optimal Segmentations

While most deep-learning based end-to-end models make greedy and possibly suboptimal local decisions to infer action segments, finding the globally optimal segmentation under the given models is desirable.

An attempt towards finding such segmentations has been made by Shi et al. (2008). They use a semi-Markov model in combination with three different feature types for segment boundaries, segment content, and interaction between neighboring segments, respectively. Action detection is formulated as a max-margin problem, which is solved by an SVM. For inference, a Viterbi-like algorithm is used to find the optimal segmentation under the model.

Hoai et al. (2011) follow a similar approach and train a multi-class SVM as segment classifier. They also use dynamic programming to find the best segmentation, which in this case, however, is not defined by maximal SVM scores of the segments. Instead, they propose to optimize in a way that segments are maximally distinguishable from other classes, thus not only optimizing for the best scoring class but also suppressing the other classes. A related method with application on detecting actions of fruit-flies (Eyjolfsdottir et al., 2014) finds that optimizing for the best segmentation – given their specific underlying action models – is prone to over-segmentation and sliding window approaches can be preferable.

Similar to these approaches, we also use a dynamic programming based algorithm to infer a globally optimal segmentation. The underlying model, however, is entirely different from the above approaches. In contrast to the SVM-based methods that rely on features capturing boundary cues, we optimize over a probabilistic framework that explicitly models long-range context and action lengths together with sophisticated visual frame or segment classifiers.

2.2.5 Explicit Context Modeling and Stochastic Grammars

The previously discussed approaches to temporal action segmentation either ignore long-range context between action instances completely or rely on TCNs to implicitly learn this context. A drawback of the latter is that they have a limited receptive field and can not learn any contextual dependencies that go beyond. Hence, for long videos with complex context dependencies, TCNs fail to capture all necessary context information.

Therefore, explicit context modeling is desirable. In an early work, Bobick and Ivanov (1998) propose to model atomic actions such as *left-leg-back* by an HMM and to manually define a context-free grammar that is used to assemble atomic actions to higher level actions. The authors evaluate their approach on hand movements of a music conductor and can successfully find bars, i.e. temporal boundaries of musical phrases, from the sequence of conducted gestures. A similar approach is proposed in İkizler and Forsyth (2008), who model atomic movements of body parts with HMMs and use regular expressions to model sequences of such atomic movements. Pirsiavash and Ramanan (2014) also use a context-free grammar to model hierarchical decompositions of actions. A parsing algorithm using finite state machines allows to efficiently infer such hierarchical temporal structures. In a related approach, Vo and Bobick (2014) propose to use a grammar that is restricted to AND and OR rules. The grammar allows to decompose high-level activities into sequences of lower-level actions and a message passing algorithm enables efficient inference over all possible decompositions. Hou et al. (2017) attempt to model context without an explicit grammar. Therefore, the input sequence is clustered into temporally connected and spatially similar subactions. Similar clusters are merged to avoid finding the same subaction class twice. The best sequence of subactions is then found by a shortest-path algorithm over the mined subaction instances. Another clustering based approach to incorporating context has been proposed by Cheng et al. (2014). In their work, a video is represented as a sequence of visual words obtained by k-means clustering. A sequence memoizer (Wood et al., 2011) is then used to learn temporal dependencies between the visual words. In terms of context modeling, Kuehne et al. (2014, 2016) are closest to our work. They approach temporal action segmentation with a classical speech recognition system. Video-frame probabilities are represented with a Gaussian mixture model and a hidden Markov model is used for the temporal progression of these features throughout a low-level action, which is the equivalent to a word in speech recognition. A context-free grammar models interdependencies between action classes and can be seen as the equivalent to a language model in speech recognition.

In our work, we also rely on context models like stochastic grammars to capture long-range dependencies between action classes. However, we use different kinds of grammars and are the first to combine them with explicit length modeling and neural networks as action models.

2.3 Weakly Supervised Learning

The need for large-scale annotated data is a limiting factor for current approaches to action segmentation. While Caba Heilbron et al. (2018) propose active learning as a solution, another way is to develop algorithms that can learn from weaker annotations. This can be either ordered action sequences without exact temporal boundaries that are cheap to annotate or labels that can be mined from subtitles and meta-tags. In the following, we discuss existing strategies for weakly supervised learning for action segmentation.

2.3.1 Weakly Supervised Learning from Structured Sequences

Compared to classical action recognition, the problem of weakly supervised learning of actions is a rather new topic. First works in this field, proposed by Laptev et al. (2008) and Marszalek et al. (2009), focus on mining training samples from movie scripts. They extract class samples based on the respective text passages and use those snippets for training without applying a dedicated temporal alignment of the action within the extracted clips. Attempts for learning action classes including temporal alignment on weakly annotated data are made by Duchenne et al. (2009). Here, movie scripts are analyzed and matched to subtitles in order to obtain temporal annotations. The task is then to temporally segment frames containing the relevant action from the background activities. The temporal alignment is thus interpreted as a clustering problem, separating temporal snippets containing the action class from the background segments. The clustering problem is formulated as a minimization of a discriminative cost function. This problem formulation is extended by Bojanowski et al. (2014), who also introduce the Hollywood Extended dataset. In their work, weak learning is formulated as a temporal assignment problem. Given a set of videos and the action order of each video, the task is to assign the respective class to each frame, thus to infer the respective action boundaries. The authors propose a discriminative clustering model using temporal ordering constraints to combine classification of each action and their temporal localization in each video clip. The Frank-Wolfe algorithm is used to solve the convex minimization problem. This method has been adopted by Alayrac et al. (2016) for unsupervised learning of task and story lines from instructional video. Another approach for weakly supervised learning from temporally ordered action lists is introduced by Huang et al. (2016). Inspired by CTC models in speech recognition (Graves et al., 2006), they use connectionist temporal classification (CTC) and introduce a visual similarity measure to prevent the CTC framework from degeneration and to enforce visually consistent segmentations. Lin et al. (2017a) use the CTC approach in combination with a statistical language model for weakly supervised video learning. However, they only infer the sequence of actions occurring in the video but no segment boundaries. A different way, also lending on the concept of speech recognition, is proposed by Kuehne et al. (2017). Here, actions are modeled by a GMM/HMM speech recognition system. The training follows an iterative scheme of pseudo ground truth generation, starting from a uniform alignment of actions to frames, and model optimization. An TCN-based idea is proposed by Ding and Xu (2018), who use a temporal convolutional feature pyramid network (TCFPN), an adaption of the encoder-decoder temporal convolutional neural networks from Lea et al. (2017), for frame-wise classification in combination with an iterative soft boundary assignment for the action sequence alignment. During training, the alignment of the sequences to the transcripts is refined by an insertion strategy, which means that one class instance is represented by multiple successive instances of the same class, allowing the temporal receptive field of the TCN to extend in the temporal domain.

2.3.2 Video-level Annotations

Other works focus on a different kind of weak supervision, where only a video-level class label is given during training without any temporal annotation. The task is then to localize all instances of this activity in the video. Here, usually pretrained networks are used to detect unseen actions in an untrimmed training set and networks are fine-tuned with respect to the detected instances. Note that classes in the datasets used for pretraining, such as UCF101 (Soomro et al., 2012), Sports1M (Karpathy et al., 2014), or Kinetics (Kay et al., 2017), can overlap with the classes to search for in the untrimmed videos, e.g. in case of the Thumos action detection task (Idrees et al., 2017). Further, the information is given which classes appear in the untrimmed videos, but not when or how often they appear. This task was first addressed by Wang et al. (2017), who select clip proposals from a set of untrimmed training videos to learn actions without exact boundary annotation. Nguyen et al. (2018) propose a combination of attention weights and temporal class activation maps for the task. In Paul et al. (2018), two-stream features are extracted and localization and classification of action instances is achieved by a multiple-instance learning loss and a co-similarity loss that encourages videos that share an action class to have similar features in the respective temporal regions. Shou et al. (2018), in contrast, use two branches, a classification branch that generates a sequence of class activations for each short video snippet, and a localization branch that predicts the segment center and width. In order to train the model, they propose a contrastive loss between tight segment boundaries and extended outer boundaries that enforces high activations in the tight boundaries and low activations in the outer boundaries. The Hide-and-Seek method proposed by Singh and Lee (2017) follows the simple idea of providing a video-level label and randomly occluding parts of the video during training, forcing the network to concentrate on all discriminative regions of the desired action class – rather than just on sparse most discriminative snippets – and therefore yielding high activation scores at temporal locations that contain the action.

Note that this kind of supervision is related to providing an unordered set of occurring actions, as we do in Chapter 8. However, the methods discussed in this section focus on localizing instances of a single class in a video, whereas we address the problem of dense action segmentation, where segments with a large number of different classes and strong long-range context dependencies need to be segmented.

2.3.3 Weakly Supervised Approaches in Other Domains

Besides the approaches focusing on weak learning of human actions, also other weak learning scenarios have been explored. Naturally, a closely related approach comes from the field of sign language recognition. Here, Koller et al. (2016) integrate CNNs with hidden Markov models to learn sign language hand shapes based on a single frame CNN model from weakly annotated data. They extend the proposed single frame model by including LSTMs for temporal correlation in Koller et al. (2017). The task addressed by Malmaud et al. (2015) is

	videos	frames	action classes	action instances
Thumos	412	2,565,495	20	5,902
Breakfast	1,720	3,590,899	48	11,267
Hollywood Extended	937	780,222	16	2,366
50 Salads	50	577,595	17	966
MPII Cooking	44	881,755	65	5,609
MPII Cooking 2	273	2,881,616	67	14,105

Table 2.1: Statistics of different datasets used for temporal action segmentation.

more speech related, trying to align recipe steps to automatically generated speech transcripts from cooking videos. They use a hybrid HMM model in combination with a CNN based visual food detector to align a sequence of instructions, e.g. from textual recipes, to a video of someone carrying out a task. Sun et al. (2015) and Gan et al. (2016) learn action classes from web images and videos retrieved by specific search queries. Gan et al. (2016) match images and video frames and use a regularization over the selected video frames to balance the matching procedure. Sun et al. (2015) iteratively train classifiers using video frames with weak video labels and web images with noisy labels as input. Exploiting the different kinds of noise in video frames and web images, relevant action frames in videos can be distinguished from background frames with similar appearance. Yan et al. (2017) use video-level tags for weakly-supervised actor-action segmentation using a multi-task ranking model to select representative supervoxels for actors and their respective actions. In the work of Sener and Yao (2018), low-level actions are learned without the low-level classes being provided as annotation. Given only a high-level activity for each video, a number of subactions is obtained from an algorithm that alternates between learning an appearance representation and a temporal subaction model. For evaluation, the obtained subaction clusters are then matched to the ground truth subactions using the Hungarian method.

2.3.4 Length Modeling for Temporal Sequences

Length models are not so widely used in action classification, but the idea has e.g. been used by Bojanowski et al. (2015) who exploit a duration prior for weakly supervised video-to-text alignment. The modeling of temporal duration has also a long tradition in the context of speech processing and has been used e.g. in Vaseghi (1995). Since then it has been used in different contexts such as general modeling in case of explicit state duration HMMs (Dewar et al., 2012) but also for speech synthesis (see e.g. Zen et al. (2007)) or the generation and decoding of temporal sequences like music (Narimatsu and Kasai, 2017).

2.4 Datasets

In this section, we describe the relevant datasets used in this thesis. An overview of the dataset statistics can be found in Table 2.1.

Kinetics. The Kinetics dataset (Kay et al., 2017) is a large-scale dataset for action recognition. It contains 400 different action classes that are single person actions as well as person-person and person-object interactions. Overall, there are 306, 245 video clips. For comparison, the widely used benchmark datasets HMDB-51 (Kuehne et al., 2011) and UCF-101 (Soomro et al., 2012) only have 6, 766 and 13, 320 clips, respectively. The clips in the Kinetics dataset are about 10 seconds long and pre-segmented, i.e. they contain a single action class each. The authors ensured that there are at least 400 clips for each action class. Being one of the largest available action recognition datasets, it is frequently used in combination with deep neural networks to extract features for other datasets.

Thumos. The Thumos dataset (Idrees et al., 2017) has become a widely used benchmark for fully supervised action detection. Originally designed for a challenge[1], it features two tasks, a video classification task and a temporal action detection task. For the first task, the 13, 320 videos and 101 action classes of UCF-101 are used as training set. Additionally, there is a validation set with 1, 010 videos, a test set with 1, 574 videos, and an optional background set with over 2, 500 videos that have been collected from YouTube. For the temporal action detection task, a subset of these videos is used. The authors selected 20 sports-related action classes out of the 101 total classes and provide temporal annotations of 200 videos from the validation set and 212 videos from the test set. This corresponds to a validation set of about 1.2 million frames and a test set of about 1.3 million frames. In total, test and validation set contain 5, 902 action instances and 6, 105 background segments. In our setups, we use the 200 temporally annotated validation videos for training and evaluate on the 212 test videos, following the official evaluation script.

In contrast to other datasets, Thumos has some special characteristics. First of all, 70.3% of all frames are labeled as background. Moreover, most videos consist of multiple action instances of a single action class. Out of the 412 videos, only 35 contain two different action classes and only one contains three different action classes. The high background portion makes it a challenging detection benchmark. Long-range context, on the other hand, is not as important.

Breakfast. The Breakfast dataset (Kuehne et al., 2014) is one of the largest available datasets for temporal action segmentation. It contains 1, 712 videos of 52 participants who prepare 10 different breakfast dishes in 18 different kitchens. Overall, this accounts for 66.5h of video data, corresponding to 3.6 million video frames. The authors defined a set of 48 action classes that are mostly shared among the 10 different dishes. All videos are densely annotated, i.e. there is background at the beginning and end of the video and in between, action instances follow each other without gaps. Overall, there are more than 11, 000 action instances in the dataset, with an average of 6.5 instances per video. For evaluation, a four-fold cross-validation is used. Each of the four dataset splits separates the videos of 13 subjects as test data, such that persons occurring in the train set do no re-occur in the test set.

The Breakfast dataset has become a standard benchmark for weakly supervised action

[1] `http://crcv.ucf.edu/THUMOS14/`

segmentation (Huang et al., 2016; Kuehne et al., 2017; Ding and Xu, 2018). It is particularly challenging because it is mainly motion driven. Many datasets have a strong appearance bias: sport activities, for instance, are frequently on grass and feature a ball or a racket, swimming is in water, and beach volleyball is on sand. Such an appearance bias can be used to leverage the performance by using strong image classification or object detection systems as outlined in Jain et al. (2015). The scenes in the Breakfast dataset, however, are kitchens. Cameras are static and objects are either small or present all the time. An investigation in Kuehne et al. (2017) shows that weakly supervised approaches particularly on Breakfast do not benefit from appearance driven CNN features. Thus, Fisher vectors of improved dense trajectories (Wang and Schmid, 2013) are still the features of choice for weakly supervised approaches on this dataset. Further, the dataset contains actions that vary greatly in length, ranging from a few seconds as for *crack egg* to several minutes as for *fry pancake*. Other actions are strongly context dependent. For instance, *stir coffee*, *stir milk*, and *stir tea* all look much alike and might be indistinguishable depending on the camera angle. Therefore, for a reliable temporal segmentation, context information is crucial.

Hollywood Extended. Hollywood Extended (Bojanowski et al., 2014) is an extension of the Hollywood2 dataset from Marszalek et al. (2009). From 69 Hollywood movies, a total of 937 video clips have been annotated with 16 different action classes. Overall, this corresponds to about $780,000$ frames and 5.9 action instances per video on average. The background ratio on this dataset is 61%. For evaluation, we use a ten-fold cross-validation where the test data for the i-th split are all videos that end with the digit $i-1$. In contrast to the Breakfast dataset, this dataset features different scenes, multiple actors, and camera motion. Although there are many single-person actions and person-person interactions such as *run* or *kiss*, some action classes are also strongly appearance driven as they include interactions with objects such as *answering phone* or *drive car*.

50 Salads. 50 Salads (Stein and McKenna, 2013) is related to the Breakfast dataset in that it contains videos of food preparation. In this case, 25 participants were asked to prepare a mixed salad twice. In order to get sufficient variations, each preparation of a salad has been done following a different order of preparation steps. The videos have been recorded with a Kinect RGB-D camera that provides a top-view on the preparation space. For additional data, accelerometers have been attached to different kitchen tools. However, in accordance with other works such as Lea et al. (2017), we only use the RGB frames. In the dataset, different levels of annotation exist. On a high level, 9 action classes have been annotated. On a more fine-grained level, the task of preparing a salad is broken down into 17 action classes each representing a preparation step such as *cut tomato* or *add salt*. We use this kind of annotation for our experiments. Moreover, there is a more fine-grained annotation available, where each of the 17 actions are further subdivided into a pre-, core-, and post-phase. Although only 50 videos have been recorded, they are quite long and the overall number of frames in the dataset is $577,595$, corresponding to $4.8h$ of video data. Overall, there are 966 annotated action instances in the dataset, leading to an average of 19.3 instances per video. For evaluation, the authors propose to use a five-fold cross-validation. In each split of the dataset, five actors (ten videos) are separated for testing, the others are used for training.

Although the dataset is rather small, the videos are long and contain a large amount of action instances. Similar to Breakfast, the actions are densely annotated and background frames only exist at the beginning and end of the video. Moreover, there is a strong context dependency in the action order.

MPII Cooking. MPII Cooking (Rohrbach et al., 2012) is a database of cooking activities. Although it only contains 44 videos, it has a considerable length of $881,755$ frames or $8h$. The videos in the dataset are of twelve participants who were asked to prepare between one and six dishes out of a set of 14 dishes. The dataset has been annotated with fine-grained action labels such as *pour* or *cut slices*. In total, there are 65 action classes and $5,609$ action instances. This is an average of 127 instances per video. Note that this includes a background class that is used to fill the gap between two actions. In contrast to the Breakfast dataset, MPII Cooking has been recorded in a single kitchen. For evaluation, the authors propose to use a seven-fold cross-validation, where each split separates a single person from the training data.

MPII Cooking 2. MPII Cooking 2 (Rohrbach et al., 2016) is an extension of the MPII Cooking dataset. The recording conditions were similar as for MPII Cooking but in this dataset, there are 30 participants and a total of 273 videos with $2,881,616$ frames. In the videos, 67 different fine-grained action classes are annotated, leading to a total of $14,105$ action instances. This corresponds to 52 instances per video on average. Since more data is available, the authors avoid a multi-fold cross-validation and define a fixed training set containing 201 videos, a validation set with 17 videos, and a test set with 42 videos. Each person only occurs in one of the sets.

Chapter 3

Preliminaries

In this chapter, we formally define the task of action recognition and temporal action segmentation. Further, we introduce the evaluation metrics used throughout the thesis and discuss the extraction of hand-crafted features that are still widely used in weakly supervised action segmentation. Since many of our methods rely on recurrent neural networks, we give a brief introduction into standard RNNs and gated recurrent units. We also show that classical pipelines based on hand-crafted features can be modeled using a simple recurrent neural network. The chapter is concluded with a review of I3D, the currently most successful deep neural network for action recognition.

Contents

3.1 Problem Description and Notation

While the focus of this thesis is on temporal action segmentation, some methods rely on results from classical action recognition. We therefore formally define both tasks and illustrate the difference between the two.

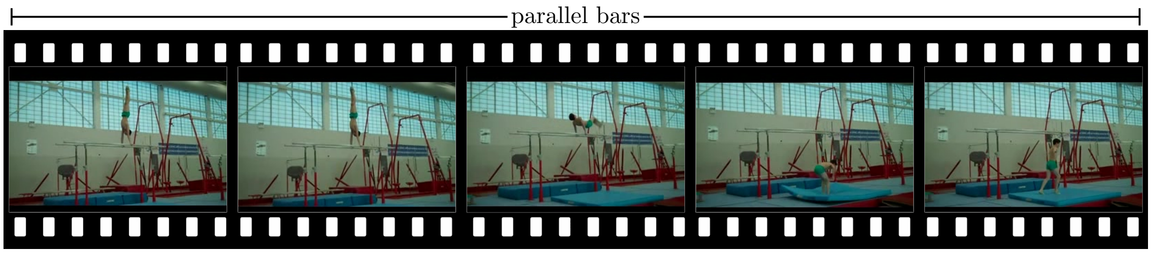


Figure 3.1: Example of a video from the UCF-101 dataset. Each video is pre-segmented and contains a single action instance from its first to its last frame, in this case `parallel bars`.

Throughout the thesis, we stick to some notation conventions. Since our methods rely on precomputed features – either handcrafted features from traditional feature extraction methods or deep features obtained from a CNN – we assume that a video is always given as a sequence of feature vectors. A sequence is written as bold symbols with the start frame as subscript and the end frame as superscript. For instance, a video sequence of T frames is written as $\mathbf{x}_1^T = (x_1, \dots, x_T)$, where each $x_t \in \mathbb{R}^D$ denotes the D-dimensional feature vector extracted at frame t of the video. In order to denote a segment of a video that ranges from frame t_1 to t_2, the notation is $\mathbf{x}_{t_1}^{t_2}$. Similarly, if we want to denote a sequence of N class labels, it is written as $\mathbf{c}_1^N = (c_1, \dots, c_N)$. The set of available classes is denoted as $\mathcal{C}$ and the space of all videos is $\mathcal{X}$.

3.1.1 Action Recognition

Action recognition is the task of assigning an action class to a pre-segmented video clip. Pre-segmented in this context means that every video contains a single action instance only, i.e. from its first to its last frame, it shows exactly one action, see Figure 3.1. Assume the video is given as a sequence of T frames $\mathbf{x}_1^T = (x_1, \dots, x_T)$. Every x_t is either the RGB frame itself, so that $x_t \in \mathbb{R}^{3 \times w \times h}$ for a video with resolution $w \times h$, or it is a D-dimensional feature vector representation $x_t \in \mathbb{R}^D$. Feature vectors can be obtained by various kinds of feature extraction methods on the frames. We introduce the most relevant ones in Section 3.3 and Section 3.7.

Further, a set of classes $\mathcal{C}$ is given, containing all possible action classes a video can be classified as. The task of action recognition can then be formalized as finding a decision rule that maps from the set $\mathcal{X}$ of all possible videos to the set of possible action classes,

$$\mathcal{R} : \mathcal{X} \mapsto \mathcal{C}. \tag{3.1}$$

Most current systems assume that the set $\mathcal{C}$ is a fixed and closed vocabulary of action classes, i.e. the set only contains action classes that occur in the training data. For inference, it is assumed that no new classes can occur. The decision rule $\mathcal{R}$ is usually learned from data in a fully supervised fashion: Assume a set of N training pairs that are video and class-label tuples $\{(\mathbf{x}_1, c_1), \dots, (\mathbf{x}_N, c_N)\}$ with $\mathbf{x}_n \in \mathcal{X}$ and $c_n \in \mathcal{C}$. Note that we omitted the subscript

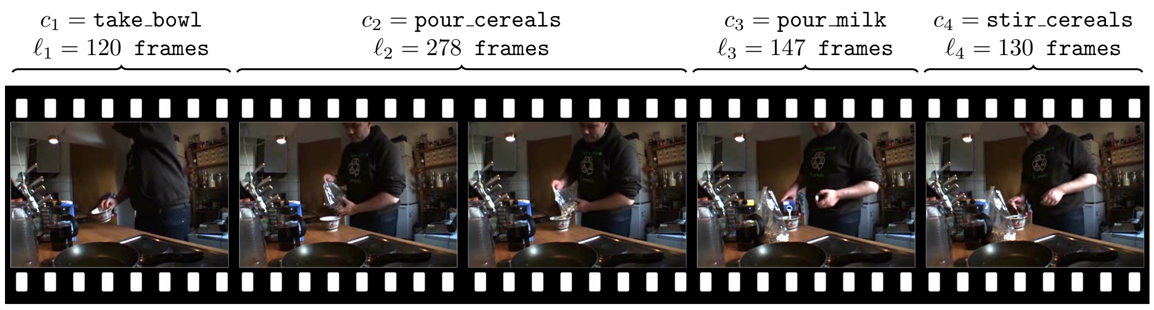


Figure 3.2: Example segmentation of a video from the Breakfast dataset. The segmentation is fully defined by the tuple $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$, with $N = 4$, class labels $\mathbf{c}_1^N = (\texttt{take\_bowl}, \texttt{pour\_cereals}, \texttt{pour\_milk}, \texttt{stir\_cereals})$ and lengths $\mathbf{l}_1^N = (120, 278, 147, 130)$.

and superscript $\bullet_1^T$ for the sequence start and end frame in $\mathbf{x}_n$ for simplicity. The decision rule $\mathcal{R}$ is typically learned using the maximum likelihood criterion,

$$\mathcal{R}_{\text{best}} = \arg\max_{\mathcal{R}} \Big\{ \sum_{n=1}^{N} \log p\big(\mathcal{R}(\mathbf{x}_n) = c_n | \mathbf{x}_n\big) \Big\}. \tag{3.2}$$

3.1.2 Temporal Action Segmentation

While videos in classical action recognition are assumed to be pre-segmented and contain exactly one action instance ranging from its first to its last frame, most practical applications require systems to be able to deal with temporally untrimmed videos. Therefore, consider a video with T frames that can contain an arbitrary number of different or repeating action instances that may be separated by non-action or background frames. The problem of temporal action segmentation is to assign an action label to every frame in the given video, where we assume that all frames not showing an action are labeled as background. In other words, the task is to generate a temporal segmentation of the video such that all frames within one segment belong to the same action.

Formally, consider an input video $\mathbf{x}_1^T = (x_1, \dots, x_T)$. We denote the unknown number of action segments in the video as N, the action class for each of these segments as a sequence $\mathbf{c}_1^N = (c_1, \dots, c_N)$, and the length of each segment as a sequence $\mathbf{l}_1^N = (\ell_1, \dots, \ell_N)$, where $1 \leq \ell_n \leq T$. The tuple $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$ then defines a full segmentation of the video: The first segment consists of frames 1 to ℓ_1 and is labeled with c_1, the second segment consists of frames $\ell_1 + 1$ to $\ell_1 + \ell_2$ and is labeled with c_2, and so on, see Figure 3.2 for an example.

Action Alignment. Action alignment is closely related to action segmentation with the only difference that the sequence of action instances $\mathbf{c}_1^N = (c_1, \dots, c_N)$ is also given during inference. While the training procedure is usually the same for both tasks, the inference for action alignment reduces to finding the segment lengths $\mathbf{l}_1^N = (\ell_1, \dots, \ell_N)$ for each of the action instances in $\mathbf{c}_1^N$.

3.2 Evaluation Metrics

We evaluate our models on test data that has not been used for training. For temporal action segmentation, there are different evaluation metrics that all try to capture the quality of the resulting temporal segmentation but differ in some details. In this section, we introduce the metrics that are relevant for the datasets we use.

Frame Accuracy. Frame accuracy measures the fraction of frames that have been correctly labeled. This metric is well suited to reflect the global quality of the segmentation. However, the metric is not sensitive to slight over-segmentations. Particularly, if a large action segment is chopped into smaller segments by a small number of wrongly labeled frames, the frame accuracy is hardly affected. On the other hand, frame accuracy is sensitive to the segment length. Segments with more frames naturally have a higher impact on the frame accuracy than short segments. The metric is the standard evaluation for the Breakfast dataset.

Mean Average Precision (mAP). Mean average precision (mAP) is the official evaluation metric for Thumos.[1] Consider a detected segment of class c and a ground truth segment of the same class. The overlap between the two segments is defined as intersection over union,

$$\mathrm{IoU} = \frac{\text{detected segment} \cap \text{ground truth segment}}{\text{detected segment} \cup \text{ground truth segment}}. \tag{3.3}$$

A segment is considered correct if its IoU with the most overlapping ground truth segment is larger than a threshold. The average precision is then computed for each class independently. Particularly, let there be K detected segments of class c. Further, assume there is a score for each of the K segments such that the segments can be ranked by this score in decreasing order. We denote the precision at cut-off k as $\mathrm{Prec}(k)$. Thus, $\mathrm{Prec}(k)$ is the ratio of correct segments in the first k top-ranked segments. Similarly, we define $\mathrm{relevant}(k)$ as an indicator function that is one if the segment ranked at position k is correct. The average precision for class c is defined as

$$\mathrm{AP}(c) = \frac{\sum_{k=1}^{K} \mathrm{Prec}(k) \cdot \mathrm{relevant}(k)}{\text{number of ground truth segments of class } c}. \tag{3.4}$$

The overall reported score is then the average over all classes,

$$\mathrm{mAP}(c) = \frac{1}{|\mathcal{C}|} \sum_{c=1}^{|\mathcal{C}|} \mathrm{AP}(c). \tag{3.5}$$

Results are reported for different IoU ratios, typically ranging from 0.1 to 0.5. In contrast to frame accuracy, this metric is sensitive to over-segmentation and does not explicitly weight segments by their lengths.

[1] Details on the evaluation protocol can be found online. `http://crcv.ucf.edu/THUMOS14/download.html`

Midpoint Hit. The midpoint hit criterion follows the same evaluation strategy as mAP with the only difference that a segment is considered correct if the midpoint of the detected segment lies within a ground truth segment of the same class. This way, it is not necessary to define an overlap threshold. However, the metric is insensitive to segment lengths. For instance, a detected segment of 100 frames length that lies within a $1,000$ frames ground truth segment is considered correct although 90% of the frames are not correct. Similarly, a detected segment of $1,000$ frames length with its midpoint within a 100 frames ground truth segment is also considered correct while again 90% of the frames are not correct. This metric is the standard evaluation criterion on MPII-Cooking and MPII-Cooking 2, where segments are dense and do not vary as much in their lengths as in other datasets like Thumos.

Jaccard Index/Intersection over Union (IoU). In this thesis, we follow the definition of the Jaccard Index used in Kuehne et al. (2017) and define it as intersection over union on a frame level. It is defined for each class as

$$\text{IoU}(c) = \frac{\text{detected frames of class } c \cap \text{ground truth frames of class } c}{\text{detected frames of class } c \cup \text{ground truth frames of class } c}. \tag{3.6}$$

The overall metric is the weighted sum over all classes, where the weight for each class is its relative frequency,

$$\text{JaccIdx} = \sum_c w_c \cdot \text{IoU}(c), \tag{3.7}$$

where

$$w_c = \frac{\text{ground truth frames of class } c}{\text{total number of frames}}. \tag{3.8}$$

Consequently, frequent classes have a larger weight than rare ones. The IoU is zero if the aligned segments do not overlap at all and one if they match the ground truth exactly. We report action segmentation results on Hollywood Extended using this metric.

Intersection over Detection (IoD). A variant of intersection over union is intersection over detection (IoD). This metric is only used for the task of action alignment when there is a one-to-one mapping between ground truth segments and detected segments. Intersection over detection is defined as

$$\text{IoD}(c) = \frac{\text{detected frames of class } c \cap \text{ground truth frames of class } c}{\text{detected frames of class } c}. \tag{3.9}$$

Same as for IoU, the overall metric is the weighted sum over all classes, where the weight for each class is its relative frequency. The metric is used for action alignment on Hollywood Extended, where the ordered action sequence is given and only the temporal segment boundaries need to be inferred. Note that an IoD of zero can not happen in practice since the first frame is always part of the first segment and thus matches the ground truth and the same holds for the last frame.

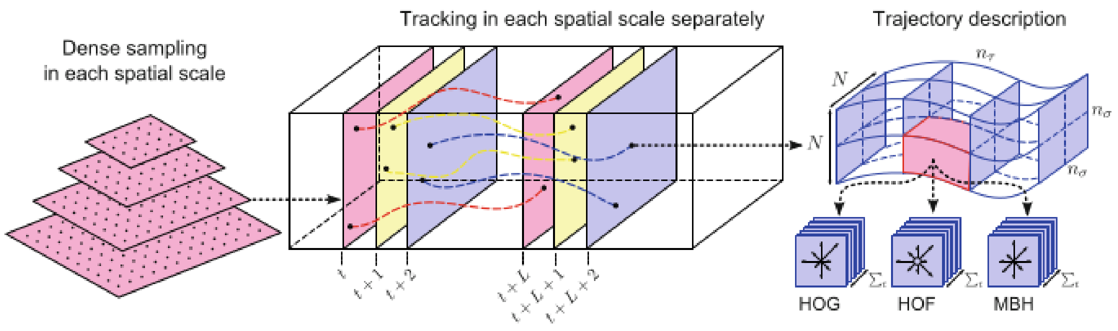


Figure 3.3: Extraction of dense trajectories from a video. Left: Dense sampling of points, middle: tracking points through the temporal volume, right: computing descriptors along a trajectory. Figure is taken from Wang et al. (2013).

3.3 Improved Dense Trajectories

Improved dense trajectories (IDTs) (Wang and Schmid, 2013) have been the de-facto state-of-the-art features in action recognition for several years before first deep architectures could outperform IDT-based systems. Nowadays, fully supervised action recognition usually works much better with deep architectures. When it comes to weakly supervised action segmentation, where no frame labels are available, however, deep architectures still fail to learn a better representation than IDTs (Kuehne et al., 2017) and most state-of-the-art methods still rely on IDTs (Huang et al., 2016; Ding and Xu, 2018). We therefore revisit improved dense trajectories here and describe a strategy to extract them on a per-frame basis. The approach basically follows three steps: first, dense sampling of points at different spatial scales, second, tracing these points through time, third, extracting feature descriptors along the found trajectories. The whole pipeline is summarized in Figure 3.3.

Dense Sampling. Points are sampled uniformly distributed over the image using a grid sampling with a space of 5 pixels between each sampled point at each spatial scale. The authors propose to remove points from homogeneous image areas using a suppression criterion proposed by Shi and Tomasi (1994) because such points are almost impossible to track accurately.

Finding Dense Trajectories. Dense trajectories rely on tracking the densely extracted points along L frames (usually $L = 15$) using the median-filtered optical flow field in its original version from Wang et al. (2013). In the improved variant of Wang and Schmid (2013), the optical flow field is corrected for camera motion by estimating a homography between two consecutive frames. Specifically, the points are tracked over L frames to form a trajectory. If for any frame at any spatial position there is no tracked point in a neighborhood of 5×5 pixels, a new trajectory is started from a point in this neighborhood if such a point exists. A trajectory starting at frame t is eventually represented as a sequence of image coordinates $\big((x_t, y_t), \dots, (x_{t+L-1}, y_{t+L-1})\big)$.

Descriptors Along Trajectories. Dense trajectories cover four types of feature descriptors along each trajectory. The first one is a description of the trajectory itself, where the trajectory is represented as a normalized sequence of image coordinate displacements. The second and third descriptors are histograms of oriented gradients (HOG) (Dalal and Triggs, 2005) and histograms of optical flow (HOF) (Laptev et al., 2008). HOG relies on histograms with eight bins each and HOF uses an additional bin for motion magnitudes close to zero. Both descriptors are extracted along a volume of $N \times N \times L$ pixels (usually $N = 32$) with the trajectory going through its center. In order to have a more fine grained resolution of the descriptors, this volume is further subdivided into $n_\sigma \times n_\sigma \times n_\tau$ subvolumes and descriptors are computed over each of these subvolumes independently. The fourth descriptor type are motion boundary histograms (MBH) (Dalal et al., 2006), i.e. derivatives of the optical flow field in x and y direction. It is computed over the same subvolumes as the HOG and HOF descriptors. Overall, this results in a 426-dimensional feature vector for each trajectory.

3.4 Feature Quantization

Video features such as dense trajectories usually result in a huge amount of trajectories starting at each frame. For practical applications, a single feature vector for each video frame or a single feature vector per video is required. Quantization methods like bag-of-words are therefore used to summarize a set of feature vectors in a single vector representation.

3.4.1 Bag-of-Words

Bag-of-Words (BoW) models, also referred to as bag-of-features, have a long tradition in computer vision and are one of the most widely used approaches for feature quantization in the pre deep learning era. In the context of action recognition, it has been the method of choice on top of many hand-crafted feature extraction methods, see e.g. Schuldt et al. (2004); Laptev et al. (2008); Ullah et al. (2010); Wang et al. (2013).

Let $\mathbf{x} = \{x_1, \dots, x_K\}$ be a set of D-dimensional feature vectors $x_k \in \mathbb{R}^D$ extracted from some video. Each x_k could for instance be a 426-dimensional trajectory feature vector as described above. Note that we slightly change the notation here: Since $\mathbf{x}$ is an unordered collection of video feature vectors that is not necessarily a temporal sequence over the frames, we write $\mathbf{x}$ instead of $\mathbf{x}_1^T$ throughout this section.

Now, assume such features are not only extracted from a single video but from a set of videos $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$. The amount of extracted feature vectors usually differs for different videos, i.e. for two videos $\mathbf{x}_i$ and $\mathbf{x}_j$ with K_i and K_j feature vectors, usually $K_i \neq K_j$.

The objective of a bag-of-words model is to quantize each observation $\mathbf{x}$ using a fixed vocabulary of M visual words, $\mathcal{V} = \{v_1, \dots, v_M\} \subset \mathbb{R}^D$. To this end, each set of features is represented as a histogram of posterior probabilities $p(v|x)$,

$$\mathcal{H}(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} h(x_k), \quad h(x_k) = \begin{pmatrix} p(v_1|x_k) \\ \vdots \\ p(v_M|x_k) \end{pmatrix}. \tag{3.10}$$

Frequently, kMeans is used to generate the visual vocabulary, i.e. the feature vectors from all available videos $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ are clustered into M clusters. The visual word v_i is then set to be the center of the i-th cluster. In this case, for each x_k, $h(x_k)$ is a unit vector with a one at position $i = \arg\min_m \|v_m - x_k\|^2$ and zeros at all other positions. The quantized representation of a set of features $\mathbf{x}$ is then the normalized histogram of closest visual words $\mathcal{H}(\mathbf{x})$.

Given the success of the traditional bag-of-words model, various extensions and improvements have been proposed over the years. These attempts range from the introduction of class-dependent vocabularies (Perronnin et al., 2006) and weighted vocabularies (Cai et al., 2010) to sparse coding (Yang et al., 2009; Wang et al., 2010) and supervised dictionary learning (Boureau et al., 2010), assuming the availability of class labels. In action recognition, however, most approaches stick to the traditional BoW model.

3.4.2 Fisher Vectors

Fisher vectors (Perronnin and Dance, 2007) are a more sophisticated feature quantization method that has been of special interest for action recognition (Wang et al., 2013; Peng et al., 2014). The overall setting is similar to the bag-of-words setting. Again, each video is given as a set of features $\mathbf{x} = \{x_1, \dots, x_K\}$, where $x_k \in \mathbb{R}^D$ is a D-dimensional feature vector. For quantization, instead of applying kMeans, consider a Gaussian mixture model of the form

$$p_\lambda(x_k) = \sum_{m=1}^{M} w_m \mathcal{N}(x_k | \mu_m, \Sigma_m), \tag{3.11}$$

where M again denotes the number of clusters or, in this context, Gaussian mixtures, $w_m \in \mathbb{R}$ is a mixture weight for the m-th component, $\mu_m \in \mathbb{R}^D$ is the mean, and the covariance matrices Σ_m are assumed to be diagonal, i.e. they can be written as a vector $\sigma_m^2 \in \mathbb{R}^D$ of variances. Assuming that a large amount of feature vectors from different videos is available, e.g. as training set, the GMM is fitted to these data or a subset of the data, respectively. Let the parameters of the Gaussian mixture model be denoted by $\lambda = \{w_m, \mu_m, \Sigma_m | m = 1, \dots, M\}$.

The underlying idea of Fisher vectors is that the average gradient

$$G_\lambda(\mathbf{x}) := \frac{1}{K} \sum_{k=1}^{K} \nabla_\lambda \log p_\lambda(x_k) \tag{3.12}$$

represents the direction and magnitude by which the GMM parameters λ would need to be changed in order to better describe the specific set $\mathbf{x}$ of video feature vectors and therefore is a good vectorized description of the set $\mathbf{x}$, regardless of the actual size of the set, i.e. regardless of the number K of feature vectors in $\mathbf{x}$.

Based on the findings of Jaakkola and Haussler (1999), Perronnin and Dance (2007) propose to normalize this gradient vector using the Fisher information matrix. Using the

Gaussian posterior probabilities

$$\gamma_k(m) := \frac{w_m \mathcal{N}(x_k|\mu_m, \sigma_m^2)}{\sum_{\tilde{m}=1}^{M} w_{\tilde{m}} \mathcal{N}(x_k|\mu_{\tilde{m}}, \sigma_{\tilde{m}}^2)}, \tag{3.13}$$

the normalized gradient vector $\mathcal{G}_\lambda(\mathbf{x})$ of a set $\mathbf{x}$ of video feature vectors is then given as

$$\mathcal{G}_\lambda(\mathbf{x}) = \big(\mathcal{G}_{\mu_m}(\mathbf{x}), \mathcal{G}_{\sigma_m}(\mathbf{x}) | m = 1, \dots, M\big), \qquad \text{where} \tag{3.14}$$

$$\mathcal{G}_{\mu_m}(\mathbf{x}) = \frac{1}{K\sqrt{w_m}} \sum_{k=1}^{K} \gamma_k(m) \Big(\frac{x_k - \mu_m}{\sigma_m}\Big), \tag{3.15}$$

$$\mathcal{G}_{\sigma_m}(\mathbf{x}) = \frac{1}{K\sqrt{2w_m}} \sum_{k=1}^{K} \gamma_k(m) \Big(\frac{(x_k - \mu_m)^2}{\sigma_m^2} - 1\Big), \tag{3.16}$$

see Sanchez et al. (2013) for a more detailed description. Note that $\mathcal{G}_{\mu_m}(\mathbf{x}) \in \mathbb{R}^D$ and $\mathcal{G}_{\sigma_m}(\mathbf{x}) \in \mathbb{R}^D$ for feature vectors $x_k \in \mathbb{R}^D$. Thus, the dimension of the Fisher vector $\mathcal{G}_\lambda(\mathbf{x})$ is $2 \cdot D \cdot M$ for a GMM with M mixture components.

Perronnin et al. (2010) argue that ℓ_2-normalization and power normalization further improve the Fisher vector, so that the set of feature vectors $\mathbf{x}$ is eventually quantized as

$$FV(\mathbf{x}) = \operatorname{sign}\big(\tilde{\mathcal{G}}_\lambda(\mathbf{x})\big) \cdot |\tilde{\mathcal{G}}_\lambda(\mathbf{x})|^{0.5}, \tag{3.17}$$

where $\tilde{\mathcal{G}}_\lambda(\mathbf{x})$ is the ℓ_2-normalized version of the original Fisher vector $\mathcal{G}_\lambda(\mathbf{x})$,

$$\tilde{\mathcal{G}}_\lambda(\mathbf{x}) = \frac{\mathcal{G}_\lambda(\mathbf{x})}{\|\mathcal{G}_\lambda(\mathbf{x})\|_2}. \tag{3.18}$$

3.4.3 Bag-of-Words and Fisher Vectors for Action Recognition

Both methods, bag-of-words and Fisher vectors, summarize a set $\mathbf{x} = \{x_1, \dots, x_K\}$ of K video feature vectors into a single vector representation, either into a normalized histogram $\mathcal{H}(\mathbf{x})$ or into the normalized Fisher vector $FV(\mathbf{x})$. In order to classify a video clip as an instance of an action class, these summarized vector representations can be fed into any classifier. Typically, a support vector machine is used. Since Fisher vectors are already high dimensional and have an underlying Fisher kernel, classification with a linear SVM is usually sufficient (Wang and Schmid, 2013). Bag-of-words, on the contrary, are rather low dimensional features and usually require the application of a kernel to improve classification performance. Given two BoW histograms $\mathcal{H}(\mathbf{x}_i)$ and $\mathcal{H}(\mathbf{x}_j)$, a kernel $\mathcal{K}\big(\mathcal{H}(\mathbf{x}_i), \mathcal{H}(\mathbf{x}_j)\big)$ is defined as the inner product

$$\mathcal{K}\big(\mathcal{H}(\mathbf{x}_i), \mathcal{H}(\mathbf{x}_j)\big) = \langle \Psi(\mathcal{H}(\mathbf{x}_i)), \Psi(\mathcal{H}(\mathbf{x}_j)) \rangle, \tag{3.19}$$

where Ψ is the feature map inducing the kernel. Note that it is not trivial to find an explicit representation of Ψ. Therefore, for training the SVM, the kernel of all pairs of training observations $\mathbf{x}_1, \dots, \mathbf{x}_N$ has to be computed. For large datasets, this can be expensive. While there is a vast variety of kernels, existing work mostly relies on a multi-channel RBF-χ^2 kernel (Wang et al., 2013).

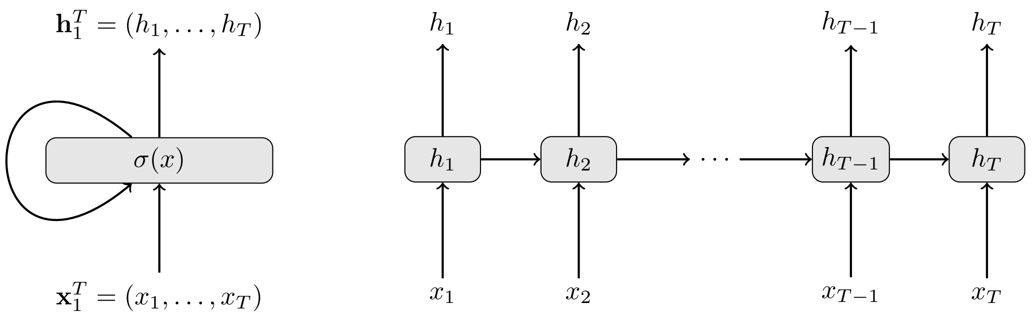

$\mathbf{h}_1^T = (h_1, \dots, h_T)$
$\sigma(x)$
$\mathbf{x}_1^T = (x_1, \dots, x_T)$
h_1
h_2
h_{T-1}
h_T
$\cdots$
x_1
x_2
x_{T-1}
x_T

Figure 3.4: Left: Schematic illustration of a vanilla RNN. Right: The same RNN unrolled over time.

3.4.4 Frame Features from IDTs with Fisher Vectors

For classical action recognition, a video level representation is sufficient. For temporal action segmentation, however, a frame level representation of video features is required. In this section, we describe how to obtain frame level features from IDTs. Assume all IDTs for a video have been extracted and reduced from 426 to 64 dimensions using PCA. In order to get frame level features for frame t, Fisher vectors are computed over all IDTs that start in the range $[t-10, t+10]$. For the computation of the Fisher vectors, a GMM with 64 mixtures is used. The resulting Fisher vector representation for each frame is then again projected to 64 dimensions using PCA, resulting in a framewise video representation $\mathbf{x}_1^T = (x_1, \dots, x_T)$ with $x_t \in \mathbb{R}^{64}$. This feature extraction is the same as proposed in Kuehne et al. (2016).

3.5 Recurrent Neural Networks

With the success of deep learning, not only convolutional neural networks but also recurrent neural networks for sequence modeling experienced increased interest. When dealing with temporal sequences of vectors as input data, neural networks need to incorporate temporal information into their layers. Early approaches on acoustic models in speech recognition therefore rely on temporal windows around each timeframe (Hinton et al., 2012; Mohamed et al., 2012), i.e. features of multiple consecutive timeframes are concatenated and then forwarded through a simple stack of fully connected layers. The drawback of such architectures is twofold. First, they require a temporal window of fixed length and can not react to different temporal dynamics in the input data. Second, the amount of required parameters grows linearly in the size of the temporal window.

Vanilla RNNs. Recurrent neural networks try to solve the problem by potentially encoding the complete sequence, not being bound to a fixed window size, and by sharing the parameters across different timeframes. A simple recurrent layer consists of forward weights $\mathbf{W}$ that map from the input dimension to the output dimension, recurrent weights $\mathbf{U}$ that realize the recurrent connection from the previous output to the current output, and a bias b. As non-linearities, frequently the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.20}$$

or the hyperbolic tangent are used. A recurrent layer then realizes the feature transformation

$$h_t = \sigma(\mathbf{W}x_t + \mathbf{U}h_{t-1} + b), \tag{3.21}$$

where x_t is the input vector at time t, h_t is the output of the hidden recurrent layer at time t, and h_{t-1} is the output of the hidden layer at the previous time step. Figure 3.4 shows an illustration of a RNN unrolled over time. Each vector h_t of the output sequence of a RNN can be seen as a high level description of the complete input sequence up to frame t. Bidirectional recurrent neural networks (Schuster and Paliwal, 1997) process the given sequence not only from beginning to end but simultaneously from end to beginning, so that each resulting output h_t is a representation of the complete input sequence with a focus on the temporal behavior around time t. The recurrent neural networks used throughout this thesis, however, are all unidirectional in order to maintain temporal causality.

Gated Recurrent Units. Vanilla RNNs tend to *forget* information from previous time steps quickly due to a repeated multiplication with $\mathbf{U}$, where the weights in the matrix are typically smaller than one to obtain a meaningful output. Ideally, RNNs should be able to learn when to focus on short range temporal dynamics and when to focus on long range temporal dynamics based on the input data. Therefore, Hochreiter and Schmidhuber (1997) propose long-short-term-memory, a recurrent network unit that is equipped with a memory cell, a *reset gate* and a *forget gate* to control how quickly or slowly past information should decay.

Here, we focus on a variant of LSTMs called gated recurrent units (GRUs) (Cho et al., 2014) which has been shown to perform on par with LSTMs (Jozefowicz et al., 2015). A GRU consists of three standard recurrent units, the *update gate* z_t, the *reset gate* r_t, and the *canditate activation* $\tilde{h}_t$. These three are then combined to the actual output activation h_t:

$$z_t = \sigma(\mathbf{W}_z x_t + \mathbf{U}_z h_{t-1} + b_z), \tag{3.22}$$

$$r_t = \sigma(\mathbf{W}_r x_t + \mathbf{U}_r h_{t-1} + b_r), \tag{3.23}$$

$$\tilde{h}_t = \tanh(\mathbf{W}_h x_t + r_t \circ (\mathbf{U}_h h_{t-1} + b_h) + b_{\tilde{h}}), \tag{3.24}$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t, \tag{3.25}$$

where x_t is the input at time t, h_t is the output of the GRU at time t, and $\circ$ denotes an element-wise multiplication. Essentially, r_t determines how high the influence of the previous activation h_{t-1} is in the new candidate activation. The update gate z_t is then used to obtain the output h_t by interpolation between the previous activation and the new candidate activation. Note that the old activation can be kept arbitrarily long if z_t is one. On the other hand, if z_t is zero, only the new candidate activation is used. It has been shown that such gating mechanisms provide quasi invariance to general time transformations (Tallec and Ollivier, 2018).

Training RNNs. The loss functions for RNNs are essentially the same that are also used for non-recurrent feed-forward networks. The widely used cross entropy loss maximizes the posterior probabilities

$$\mathcal{L} = -\sum_{n=1}^{N} \log p(c_n | x_n), \tag{3.26}$$

where n denotes the number of training samples and (x_n, c_n) is a tuple of a feature vector x_n and the corresponding class label c_n. When the input data are temporal sequences, however, the probability of a class c_t – or more generally the output h_t of a RNN – at time t of the sequence does not only depend on the feature vector x_t at frame t but on all preceding inputs, too, as can be seen in Figure 3.4 (right). Consequently, the posterior probability that is actually to be maximized for each frame in each sequence is $p(c_t|\mathbf{x}_1^t)$. While the gradient for non-recurrent neural networks is computed on a per-frame basis using backpropagation (Rumelhart et al., 1986), for recurrent neural networks, the gradient computation not only requires to backpropagate through all network layers but also through all time steps, which is known as backpropagation through time (BPTT) (Werbos, 1988).

3.6 Closing the Gap: A BoW Equivalent Neural Network

Like classical models such as bag-of-words, recurrent neural networks can be used to summarize a sequence of input vectors into a single vector representation. In this section, we show how to convert the traditional BoW model into a recurrent neural network. We also show that kernels that are frequently applied in combination with SVMs can be implemented as neural network layers. As a result, the bag-of-words pipeline can be expressed as a simple recurrent neural network with some minor approximations. We have first introduced this BoW-equivalent neural network in Richard and Gall (2015) and extended it in Richard and Gall (2017).

3.6.1 Conversion of BoW into a Neural Network

Bag-of-words models use kMeans clustering to quantize input features. The result of kMeans can be seen as a mixture distribution describing the structure of the input space. Such distributions can also be modeled with neural networks. In the following, we propose a transformation of the bag-of-words model into a neural network.

The nearest visual word $\hat{v} = \arg\min_m \|x - v_m\|^2$ for a feature vector x can be seen as the maximizing argument of the posterior form of a Gaussian distribution,

$$\begin{aligned} p_{\text{KM}}(v_m|x) &= \frac{p(v_m)p(x|v_m)}{\sum_{\tilde{m}} p(v_{\tilde{m}})p(x|v_{\tilde{m}})} && (3.27) \\ &= \frac{\exp\big(-\frac{1}{2}(x - v_m)^\intercal (x - v_m)\big)}{\sum_{\tilde{m}} \exp\big(-\frac{1}{2}(x - v_{\tilde{m}})^\intercal (x - v_{\tilde{m}})\big)}, && (3.28) \end{aligned}$$

assuming a uniform prior $p(v_m)$ and a normal distribution $p(x|v_m) = \mathcal{N}(x|v_m, \mathbf{I})$ with mean v_m and unit variance. Using maximum approximation, i.e. shifting all probability mass to the most likely visual word, a probabilistic interpretation for kMeans can be obtained:

$$\hat{p}_{\mathrm{KM}}(v_m|x) = \begin{cases} 1, & \text{if } v_m = \arg\max_{\tilde{m}} p_{\mathrm{KM}}(v_{\tilde{m}}|x), \\ 0, & \text{otherwise.} \end{cases} \tag{3.29}$$

Inserting $\hat{p}_{\mathrm{KM}}(v_m|x)$ into the histogram equation (3.10) is equivalent to counting how often each visual word v_m is the nearest representative for the feature vectors $x_1, \dots, x_T$ of a sequence $\mathbf{x}_1^T$.

Now, consider a single-layer neural network with input $x \in \mathbb{R}^D$ and M-dimensional softmax output that defines the posterior distribution

$$p_{\mathrm{NN}}(v_m|x) := \mathrm{softmax}_m(\mathbf{W}x + b) \tag{3.30}$$

$$= \frac{\exp\big(\sum_d w_{m,d}x_d + b_m\big)}{\sum_{\tilde{m}} \exp\big(\sum_d w_{\tilde{m},d}x_d + b_{\tilde{m}}\big)}, \tag{3.31}$$

where $\mathbf{W} \in \mathbb{R}^{M \times D}$ is a weight matrix and $b \in \mathbb{R}^M$ the bias. With the definition

$$\mathbf{W} = (v_1 \dots v_M)^\intercal, \tag{3.32}$$

$$b = -\frac{1}{2}(v_1^\intercal v_1 \dots v_M^\intercal v_M)^\intercal, \tag{3.33}$$

an expansion of Equation (3.28) reveals that

$$p_{\mathrm{NN}}(v_m|x) = \frac{\exp\big(-\frac{1}{2}v_m^\intercal v_m + v_m^\intercal x\big)}{\sum_{\tilde{m}} \exp\big(-\frac{1}{2}v_{\tilde{m}}^\intercal v_{\tilde{m}} + v_{\tilde{m}}^\intercal x\big)} = p_{\mathrm{KM}}(v_m|x). \tag{3.34}$$

A recurrent layer without bias and with unit matrix as weights for both the incoming and recurrent connection is added to realize the summation over the posteriors $p_{\mathrm{NN}}(v_m|x)$ for the histogram computation, cf. Equation (3.10). The histogram normalization is achieved using the activation function

$$\sigma_t(z) = \begin{cases} z & \text{if } t < T, \\ \frac{1}{T}z & \text{if } t = T. \end{cases} \tag{3.35}$$

Given an input sequence $\mathbf{x}_1^T = (x_1, \dots, x_T)$ of length T, the output of the recurrent layer is

$$\sigma_T\Big(h(x_T) + \sigma_{T-1}\big(h(x_{T-1}) + \sigma_{T-2}(h(x_{T-2}) + \dots)\big)\Big)$$

$$= \frac{1}{T}\sum_{t=1}^{T} h(x_t) = \mathcal{H}(\mathbf{x}_1^T), \tag{3.36}$$

where $h(x_t)$ denotes the M-dimensional vector of posterior probabilities $p_{\mathrm{NN}}(v_m|x_t)$.

So far, the neural network computes the histograms $\mathcal{H}(\mathbf{x}_1^T)$ for given visual words $v_1, \dots, v_M$. In order to train the visual words discriminatively and from scratch, an additional softmax layer with $|\mathcal{C}|$ output units is added to model the class posterior distribution

$$p(c|\mathcal{H}(\mathbf{x}_1^T)) = \mathrm{softmax}_c(\widetilde{\mathbf{W}}\mathcal{H}(\mathbf{x}_1^T) + \widetilde{b}). \tag{3.37}$$

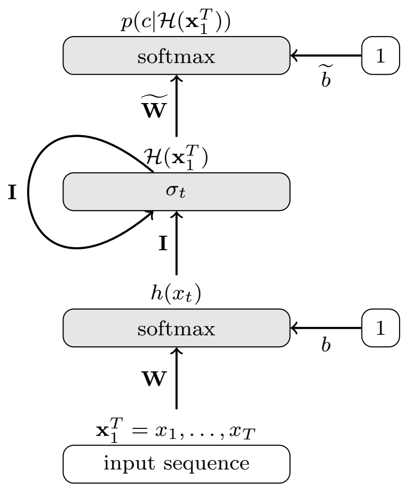


Figure 3.5: Neural network encoding the bag-of-words model. The output layer is discarded after training and the histograms from the recurrent layer can be used for classification in combination with a support vector machine.

It acts as a linear classifier on the histograms and allows for the application of standard neural network optimization methods for the joint estimation of the visual words and classifier weights. Once the network is trained, the softmax output layer can be discarded and the output of the recurrent layer is used as histogram representation. The complete neural network is depicted in Figure 3.5.

Note the difference of this method to other supervised learning methods like the restricted Boltzman machine of Goh et al. (2012). Usually, each feature vector x_t extracted from a video gets assigned the class of the respective video. Then, the codebook is optimized to distinguish the classes based on the representations $h(x_t)$. For the actual classification, however, a global video representation $\mathcal{H}(\mathbf{x}_1^T)$ is used. In our approach, on the contrary, the codebook is optimized to distinguish the classes based on the final representation $\mathcal{H}(\mathbf{x}_1^T)$ directly rather than on an intermediate quantity $h(x_t)$.

3.6.2 Equivalence Results

There is a close relation between single-layer neural networks and Gaussian models (Macherey and Ney, 2003; Heigold et al., 2007). We consider the special case of kMeans here. Following the derivation in the previous section, the kMeans model can be transformed into a single-layer neural network. For the other direction, however, the constraint that the bias components are inner products of the weight matrix columns (see Equations (3.32) and (3.33)) is not met when optimizing the neural network parameters. In fact, the single-layer neural network is equivalent to a kMeans model with non-uniform visual word priors $p(v_m)$. While the transformation from a kMeans model to a neural network is defined by Equation (3.32) and Equation (3.33), the transformation from the neural network model with weights $\mathbf{W}$ and bias

b to a kMeans model is given by

$$v_m = (w_{m,1} \dots w_{m,D})^\intercal, \tag{3.38}$$

$$p_{\text{NN}}(v_m) = \frac{\exp\left(b_m + \frac{1}{2} v_m^\intercal v_m\right)}{\sum_{\tilde{m}} \exp\left(b_{\tilde{m}} + \frac{1}{2} v_{\tilde{m}}^\intercal v_{\tilde{m}}\right)}, \tag{3.39}$$

where v_m are the visual words or the codebook learned by the neural network and $p_{\text{NN}}(v_m)$ is a prior on the visual words that is implicitly learned rather than being assumed uniform as for the case of kMeans. The neural network representation from Figure 3.5 is thus a slightly more general model than the original BoW model.

3.6.3 Encoding Kernels in the Neural Network

So far, the recurrent neural network is capable of computing bag-of-words like histograms that are then used in a support vector machine in combination with a kernel. In this section, we show how to incorporate the kernel itself into the neural network.

Consider the histograms h_1 and h_2 of two input sequences $\mathbf{x}_1^{T_x}$ and $\mathbf{y}_1^{T_y}$,

$$h_1 = \mathcal{H}(\mathbf{x}_1^{T_x}), \quad h_2 = \mathcal{H}(\mathbf{y}_1^{T_y}), \quad h_1, h_2 \in \mathbb{R}^M. \tag{3.40}$$

The kernel $\mathcal{K}(h_1, h_2)$ induced by its underlying feature map Ψ is then the inner product

$$\mathcal{K}(h_1, h_2) = \langle \Psi(h_1), \Psi(h_2) \rangle. \tag{3.41}$$

In order to avoid the explicit computation of the kernel, we used the findings of Vedaldi and Zisserman (2012), who provide an explicit (approximate) representation for additive homogeneous kernels. A kernel is called *additive* if

$$\mathcal{K}(h_1, h_2) = \sum_{m=1}^{M} k(h_{1,m}, h_{2,m}), \tag{3.42}$$

where $k : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$ is a subkernel induced by a feature map ψ. The subkernel k is *homogeneous* if

$$k(\alpha h_{1,m}, \alpha h_{2,m}) = \alpha k(h_{1,m}, h_{2,m}). \tag{3.43}$$

Vedaldi and Zisserman (2012) analyze such kinds of kernels using their Fourier spectrum and derive the feature map approximation ψ as

$$[\psi(x)]_j = \begin{cases} \sqrt{\kappa(0) x L} & j = 0, \\ \sqrt{2\kappa(\frac{j+1}{2} L) x L} \cos(\frac{j+1}{2} L \log x) & j \text{ odd}, \\ \sqrt{2\kappa(\frac{j}{2} L) x L} \sin(\frac{j}{2} L \log x) & j \text{ even}, \end{cases} \tag{3.44}$$

where each j is a component of the vector valued function ψ, κ is a function dependent on the kernel, L is a sampling period, and the number of samples is defined by n with $0 \leq j \leq 2n$.

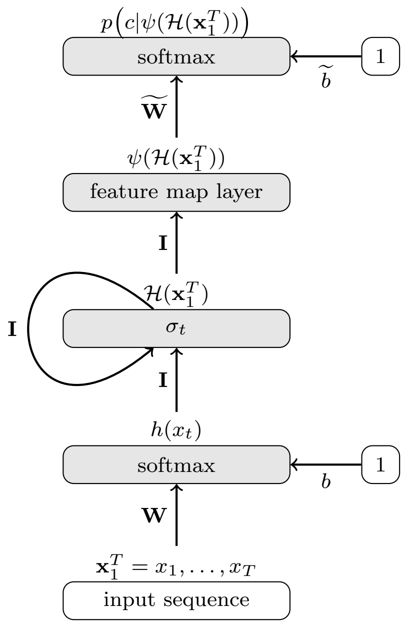


Figure 3.6: The BoW-equivalent neural network. The first softmax layer followed by the recurrent averaging layer is a soft approximation of the kMeans feature quantization. The feature map layer and the second softmax layer replace the kernel and SVM, respectively.

Recall that $\psi(x)$ is the feature map of the one-dimensional subkernel k. The final approximate feature map Ψ for a kernel as in Equation (3.42) therefore is a concatenation of the $2n+1$ elements of ψ for each of the M components of the given histogram.

The function $[\psi(x)]_j$ is continuously differentiable on the non-negative real numbers and the derivative is given by

$$\frac{\partial[\psi(x)]_j}{\partial x} = \begin{cases} [\psi(x)]_0\gamma(x) & j = 0, \\ ([\psi(x)]_{j+1}(j+1)L + [\psi(x)]_j)\gamma(x) & j \text{ odd}, \\ ([\psi(x)]_{j-1}jL + [\psi(x)]_j)\gamma(x) & j \text{ even}, \end{cases} \tag{3.45}$$

where

$$\gamma(x) = \frac{\kappa(0)L}{2[\psi(x)]_0^2}. \tag{3.46}$$

Since the widely used RBF-χ^2 kernel is not additive and homogeneous, we replace it by a version of the χ^2 kernel that meets these requirements,

$$k(h_{1,m}, h_{2,m}) = 2 \cdot \frac{h_{1,m}h_{2,m}}{h_{1,m} + h_{2,m}}, \tag{3.47}$$

which can be approximated using Equation (3.44) with the kernel-specific function

$$\kappa_\chi(\lambda) = \operatorname{sech}(\pi\lambda), \tag{3.48}$$

where sech is the hyperbolic secant.

Considering that the histograms $\mathcal{H}(\mathbf{x}_1^T)$ computed by the neural network are non-negative and that Equation (3.44) is fully differentiable on non-negative inputs, ψ can be implemented as a layer in a neural network. Adding such a feature map layer between the recurrent layer and the softmax output allows to represent the bag-of-words pipeline including linear classifier and kernel computations completely in a single neural network, cf. Figure 3.6.

To illustrate that this modification of the neural network is in fact sufficient to model a support vector machine with a non-linear kernel, consider a simple two class problem. The classification rule for the support vector machine is then

$$r_{\text{SVM}}(\mathcal{H}(\mathbf{x}_1^T)) = \operatorname{sign}\Big(\sum_{i=1}^{I} \alpha_i y_i \mathcal{K}(\mathcal{H}([\mathbf{x}^{(i)}]_1^{T_i}), \mathcal{H}(\mathbf{x}_1^T)) + b \Big) \tag{3.49}$$

with I support vectors $\mathcal{H}([\mathbf{x}^{(i)}]_1^{T_i})$ and coefficients α_i as well as labels $y_i \in \{-1, 1\}$. Defining

$$\mathbf{w}_c = \sum_{i=1}^{I} \alpha_i y_i \Psi(\mathcal{H}([\mathbf{x}^{(i)}]_1^{T_i})) \tag{3.50}$$

allows to simplify the decision rule to

$$r_{\text{SVM}}(\mathcal{H}(\mathbf{x}_1^T)) = \operatorname{sign}\big(\langle \mathbf{w}_c, \Psi(\mathcal{H}(\mathbf{x}_1^T))\rangle + b\big). \tag{3.51}$$

As can be seen from this equation, the decision rule is an inner product of a weight vector and the feature map $\psi(\mathcal{H}(\mathbf{x}_1^T))$. This is the same operation that is performed in the neural network, with the exception that the softmax layer outputs normalized probabilities rather than unnormalized scores. The class with the maximal probability or score, however, is the same in both cases. So, the decision of the support vector machine and the decision of the neural network are the same if the same weights and bias are used. Still, in contrast to the support vector machine, the neural network is trained according to the cross-entropy criterion using unconstrained optimization. Thus, due to different loss functions, the neural network usually differs from the model obtained with a support vector machine, but it is possible to convert the SVM model to an equivalent neural network model.

Note that the approximate feature map increases the dimension and, thus, also the number of parameters, depending on the number of samples. If the histograms are of dimension M, the output of the feature map layer is of dimension $M \cdot (2n + 1)$. In practice, $n = 2$ already works well (Vedaldi and Zisserman, 2012). We will use this neural network based version of the bag-of-words model in Chapter 5 in the context of fully supervised action segmentation.

3.7 Deep Learning for Action Recognition

Deep learning has greatly pushed performance in action recognition in the recent years. The most successful architectures are variants of the two-stream network from Simonyan and Zisserman (2014), which consist of two CNNs that are forwarded in parallel, see Figure 3.7. One

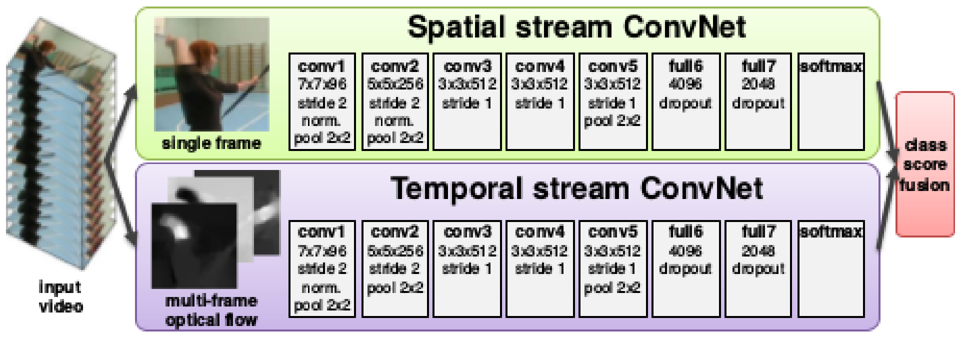


Figure 3.7: The two-stream architecture. The RGB frames of the video and stacked optical flow fields are forwarded through two deep CNN networks and their results are fused before the classification layer. Figure is taken from Simonyan and Zisserman (2014).

stream processes single RGB frames in order to extract appearance features, the other processes a stack of optical flow frames around the current frame in order to model motion. The output of both networks is fused in the end and a common classification score is computed. In general, any 2D convolutional network architecture can be used as a two-stream network. While the original version is based on an architecture proposed by Chatfield et al. (2014) for image classification, recent improvements such as the work of Feichtenhofer et al. (2017b) use more sophisticated architectures like a ResNet (He et al., 2016).

The current state-of-the-art architecture is I3D (Carreira and Zisserman, 2017), a version of two-stream networks inflated to three dimensions, i.e. not only working in the spatial domain but also in the temporal domain. The underlying architecture of I3D is an Inception-v1 network (Szegedy et al., 2015). Starting from a network that is pretrained on the Imagenet dataset (Russakovsky et al., 2015), all $N \times N$ kernels are inflated to $N \times N \times N$ kernels by copying the original 2D kernels N times and rescaling them with the factor $1/N$. Since reasonable pooling strides are different between the spatial and the temporal domain, temporal pooling is omitted in the first two max-pooling layers of Inception-v1. In the last average-pooling layer, the temporal kernel size is 2 as opposed to 7 in the spatial domain. The other max-pooling layers have the same spatial and temporal stride, see Carreira and Zisserman (2017) for details.

Feature Extraction from I3D.

In some of our experiments, we extract video features using an I3D network that is trained on the Kinetics dataset which has been proposed in Kay et al. (2017). In a first step, the optical flow of the input video is computed using the TV-L1 algorithm from Zach et al. (2007). Then, the resulting optical flow fields and the RGB frames are scaled such that the smaller side has a length of 256 pixels. In order to obtain a feature vector for each frame, a spatio-temporal volume of size $21 \times 224 \times 224$ is cropped such that the spatial 224×224 part completely lies within the video frame size and the frame we currently extract features for is in the center of the temporal window. This spatio-temporal volume is then forwarded through the pretrained I3D network. We discard the last convolutional layer and extract the 1024-dimensional features for both RGB and optical flow stream instead, resulting in a 2048-dimensional feature vector for each frame.

Chapter 4

A General Framework for Temporal Action Segmentation

In this chapter, we introduce a general framework for temporal action segmentation. The framework has a simple but sound mathematical motivation and allows to model visual information of the video frames, a length prior on the action classes, and a context model independently. In contrast to existing approaches for temporal action segmentation, a globally optimal segmentation given the individual models is computed and no local or greedy decisions are made during inference. Moreover, the framework can be applied to various tasks related to temporal action modeling such as action segmentation and action alignment. It is also applicable under varying training conditions. In the following chapters, the framework is used for fully supervised training where full frame-level annotation is required, in several weakly supervised approaches that do not need explicit temporal annotation, and in semi-supervised learning where frames are annotated sparsely.

Contents

4.1 Introduction

Given the success of recent architectures for action recognition on pre-segmented video clips, there is an increased interest in temporal action recognition over the recent years. Not only action specific deep neural networks such as the two-stream network of Simonyan and Zisserman (2014) and the temporal segment networks of Wang et al. (2016) have inspired recent works on temporal action segmentation. Some methods also borrow ideas from the field of object detection, relying on proposal networks (Lin et al., 2018) or similar multistage approaches (Shou et al., 2016), or adapt object detection frameworks directly into the one-dimensional domain of temporal action segmentation (Chao et al., 2018). Early works on temporal action segmentation mostly apply sliding window approaches (Rohrbach et al., 2012;

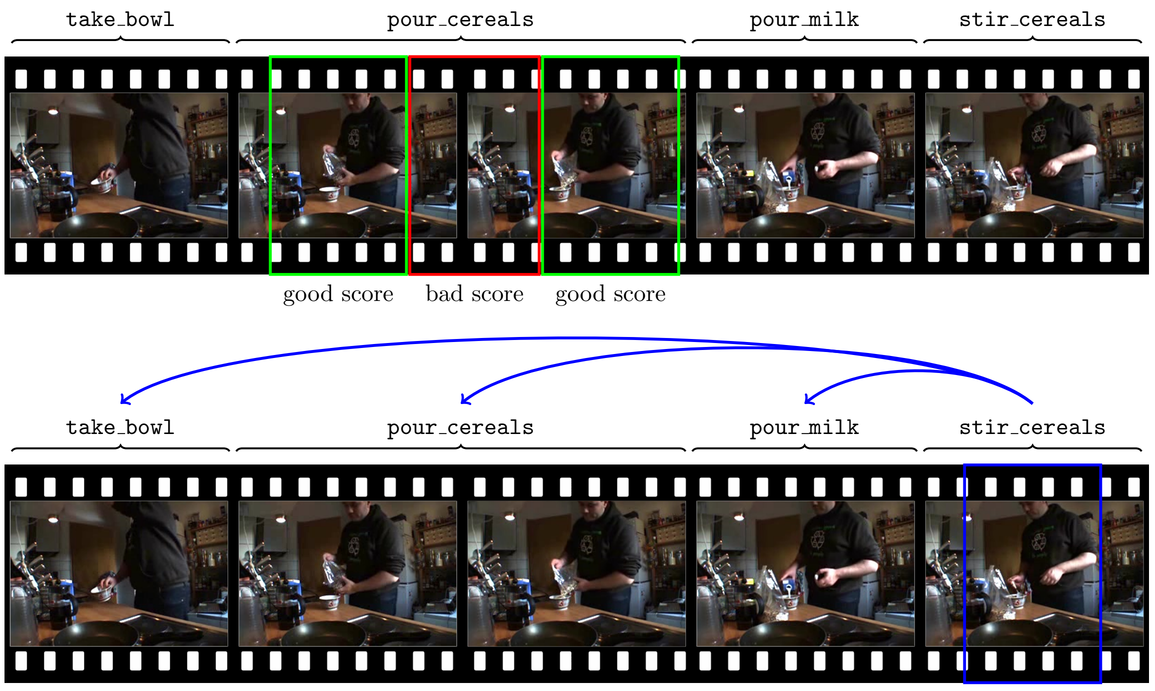


Figure 4.1: Top: Sliding windows are prone to over-segmentation as not all windows in a long action might have a good score for that particular action. A length prior can prevent or penalize such over-segmentation. Bottom: Sliding window approaches are not context aware which can lead to false classifications. For instance, `stir_cereals` might look similar to `stir_coffee` and only the context gives a clear hint which of the two options is more likely.

Oneata et al., 2014). Even recent deep learning based architectures such as the ones proposed by Zhao et al. (2017) or Wang et al. (2017) are frequently based on segment proposals or clip sampling, which are related to sliding windows. Both, sliding windows and other segment proposal methods, however, have some considerable disadvantages. Long action segments, for instance, are likely to be over-segmented since not all windows or proposals have a good score throughout the duration of the action. Taking a length prior into account can improve the results by avoiding typically long actions to be cut into multiple segments. Moreover, sliding windows and segment proposals are usually context agnostic. When hypothesizing an action class at some position in the video, only the frames in the current temporal region are taken into account. Context of previously hypothesized classes is not used at all, see Figure 4.1 for an illustration of these effects.

Furthermore, sliding windows are usually used with different temporal lengths and moved over the video with different overlap ratios. This requires some postprocessing, e.g. a non-maximum suppression to obtain a set of non-overlapping windows that form the final segmentation. Such steps, however, only find local optima. The globally best segmentation of the video may be different due to length, context, and appearance dependencies within the video.

In this chapter, we propose a general framework for temporal action segmentation that incorporates length, context, and video frame information. The framework allows to find a globally optimal segmentation, i.e. the best possible segmentation of the complete video given the explicit models. Moreover, in contrast to sliding window approaches, it is context aware and features an explicit length modeling.

The proposed framework is used in different approaches to temporal action segmentation and detection throughout this thesis, ranging from fully supervised settings where frame-level annotation is provided to weakly supervised settings where only the occurring actions are known without any temporal annotation. On a variety of datasets with different characteristics and on various temporal segmentation tasks, we prove that this general framework can yield considerable improvements over related methods.

4.2 A Generic Probabilistic Model

Following the notation introduced in Section 3.1.2, let $\mathbf{x}_1^T$ denote a video with T frames, $\mathbf{c}_1^N$ a sequence of N actions, and $\mathbf{l}_1^N$ a sequence of N lengths. In order to determine the best value $\hat{N}$ of unknown segments in the video, the optimal sequence of action labels $\hat{\mathbf{c}}_1^{\hat{N}}$, and the optimal sequence of lengths for each segment $\hat{\mathbf{l}}_1^{\hat{N}}$, the posterior probability of the segmentation given the video is to be maximized,

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{c}_1^N, \mathbf{l}_1^N | \mathbf{x}_1^T) \Big\}. \tag{4.1}$$

Using Bayes rule, this probability can be factorized into three parts,

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) \cdot p(\mathbf{l}_1^N | \mathbf{c}_1^N) \cdot p(\mathbf{c}_1^N) \Big\}. \tag{4.2}$$

Note that $p(\mathbf{x}_1^T)$ has been omitted in the factorization as it does not affect the maximizing arguments.

The first term, $p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N)$, is the probability of the frame representations $x_1, \dots, x_T$ being generated by the hypothesized segmentation $(\mathbf{c}_1^N, \mathbf{l}_1^N)$. We call it the *visual model.* Since the second term $p(\mathbf{l}_1^N | \mathbf{c}_1^N)$ models the probabilities of the length of each segment, we refer to it as *length model.* The likelihood of an action sequence $c_1, \dots, c_N$ is given by $p(\mathbf{c}_1^N)$ and we call it the *context model.*

4.3 Viterbi Decoding

In order to compute the $\arg\max$ from Equation (4.2), we use variants of the Viterbi algorithm (Viterbi, 1967). The Viterbi algorithm has a long tradition in natural language processing, particularly in automatic speech recognition, see e.g. Jurafsky et al. (1995) or Ney and Ortmanns (1999), and relies on dynamic programming to find the most likely sequence of states – the *Viterbi path* – in a temporal sequence. We present a simple version of Viterbi decoding here that serves as a basis for different variations used throughout this thesis.

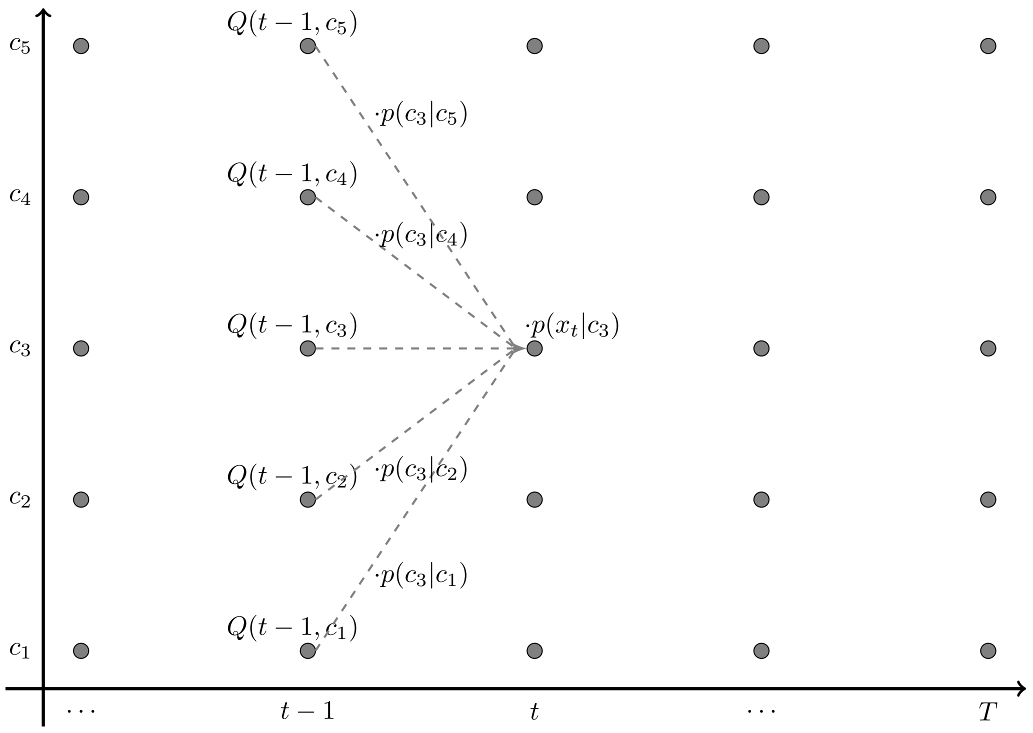


Figure 4.2: Computation of the recursive function Q via dynamic programming. At frame t, either the current segment is continued or a new action segment with class c starts. The best score at this position is the maximum of all scores at position $t-1$ multiplied by the frame score at position t. If a new segment with another class starts at frame t, the context model probability also needs to be multiplied.

4.3.1 Computing the Score of the Viterbi Path

Consider a simplification of the general model from Equation (4.2), where we omit the length model and use a simplified visual and context model. For the context model, assume that the probability for observing an action label c_n only depends on the previous label c_{n-1}, i.e.

$$p(\mathbf{c}_1^N) = \prod_{n=1}^{N} p(c_n|c_{n-1}). \tag{4.3}$$

For the visual model, let $n(t)$ denote the segment number n a frame t belongs to under the segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$. Then, $c_{n(t)}$ denotes the action label assigned to frame t. Assuming conditional independence of the frames and independence of the frames and segment lengths, the simplified visual model is

$$p(\mathbf{x}_1^T|\mathbf{c}_1^N, \mathbf{l}_1^N) = \prod_{t=1}^{T} p(x_t|c_{n(t)}). \tag{4.4}$$

Note that the lengths $\mathbf{l}_1^N$ occur implicitly in $n(t)$. Specifically,

$$n(t) = k \quad \text{if and only if} \quad \sum_{n=1}^{k-1} \ell_n < t \leq \sum_{n=1}^{k} \ell_n. \tag{4.5}$$

We call the simplified visual model a *framewise visual model* and the simplified context model a *bigram context model*.

The general model from Equation (4.2) then simplifies to

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N,\mathbf{c}_1^N,\mathbf{l}_1^N} \Big\{ \prod_{t=1}^{T} p(x_t|c_{n(t)}) \cdot \prod_{n=1}^{N} p(c_n|c_{n-1}) \Big\}. \tag{4.6}$$

With a slight abuse of notation, the segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$ can equivalently be written as a sequence of T class labels $\mathbf{c}_1^T$, where c_t is the label assigned to frame t. Equation (4.6) can then be written as

$$\hat{\mathbf{c}}_1^T = \operatorname*{arg\,max}_{\mathbf{c}_1^T} \prod_{t=1}^{T} \psi(t, c_t, c_{t-1}) \tag{4.7}$$

with the definition

$$\psi(t, c, c') = \begin{cases} p(x_t|c), & \text{if } n(t-1) = n(t), \\ p(x_t|c) \cdot p(c|c'), & \text{otherwise.} \end{cases} \tag{4.8}$$

Note that the introduction of $\psi(t)$ allows to formulate the term in the $\arg\max$ of Equation (4.7) as a product over all timeframes. For now, let the task be to only find the probability of the most likely sequence $\hat{\mathbf{c}}_1^T$, so the $\arg\max$ can be replaced with a max, providing the maximal probability but not the maximizing segmentation. Considering the input sequence only up to frame t, Equation (4.7) can be factorized as

$$\max_{\mathbf{c}_1^t} \prod_{\tau=1}^{t} \psi(\tau, c_\tau, c_{\tau-1}) = \max_{\substack{c,c':\\ c=c_t}} \Big[\max_{\substack{\mathbf{c}_1^{t-1}:\\ c_{t-1}=c'}} \prod_{\tau=1}^{t-1} \psi(\tau, c_\tau, c_{\tau-1}) \Big] \cdot \psi(t, c, c'). \tag{4.9}$$

Defining

$$Q(t, c) = \max_{\substack{\mathbf{c}_1^t\\ c_t=c}} \prod_{\tau=1}^{t} \psi(\tau, c_\tau, c_{\tau-1}) \tag{4.10}$$

and comparing Equation (4.10) with Equation (4.9) reveals the recursive nature of $Q(t, c)$: the left hand side of Equation (4.9) is $\max_{c_t} Q(t, c_t)$ and on the right hand side, the term in brackets is $Q(t-1, c')$. Note that $Q(t, c)$ is the score of the best path ending at frame t in class c. The score of the best path over the complete video sequence is therefore given as $\max_c Q(T, c)$, i.e. the score at the last frame T of the sequence ending the maximizing action class c.

In order to efficiently compute the values of $Q(t,c)$ for all time/label pairs (t,c), we exploit its recursive nature and consider the two cases occurring within the function ψ separately. In the first case, when $n(t-1) = n(t)$, the segment does not change between frame $t-1$ and frame t. Then, the class c is the same as in the previous frame and the maximization over the preceding class c' can be omitted. Using the factorization from Equation (4.9) together with the first case in the function ψ, we have

$$\begin{aligned} Q(t,c) &= \max_{\substack{\mathbf{c}_1^t \\ c_t = c}} \prod_{\tau=1}^{t} \psi(\tau, c_\tau, c_{\tau-1}) && (4.11)\\ &= \max_{c'} \Big[\max_{\substack{\mathbf{c}_1^{t-1}: \\ c_{t-1}=c'}} \prod_{\tau=1}^{t-1} \psi(\tau, c_\tau, c_{\tau-1}) \Big] \cdot \psi(t, c, c') && (4.12)\\ &= \max_{c'} Q(t-1, c) \cdot p(x_t|c) && (4.13)\\ &= Q(t-1, c) \cdot p(x_t|c). && (4.14) \end{aligned}$$

In the second case, a new action segment starts at time t. Now, using the factorization from Equation (4.9) together with the second case in the function ψ, we have

$$\begin{aligned} Q(t,c) &= \max_{\substack{\mathbf{c}_1^t \\ c_t = c}} \prod_{\tau=1}^{t} \psi(\tau, c_\tau, c_{\tau-1}) && (4.15)\\ &= \max_{c'} \Big[\max_{\substack{\mathbf{c}_1^{t-1}: \\ c_{t-1}=c'}} \prod_{\tau=1}^{t-1} \psi(\tau, c_\tau, c_{\tau-1}) \Big] \cdot \psi(t, c, c') && (4.16)\\ &= \max_{c'} \big\{ Q(t-1, c') \cdot p(x_t|c) \cdot p(c|c') \big\}. && (4.17) \end{aligned}$$

Hence, the score $Q(t,c)$ can be computed from all possible scores $Q(t-1,c')$ of segments ending at frame $t-1$ in class c'. Since a new segment starts at frame t, the context probability of transitioning from the old action class c' to the new action class c has to be multiplied with the old score $Q(t-1,c')$ along with the framewise visual model probability. The recursive computation of Q is illustrated in Figure 4.2. Note that for the case of staying in the same class c, continuing the current segment (first case in ψ) always leads to a better score than starting a new segment (second case in ψ) with the same class as this would require an additional multiplication with $p(c|c)$.

A pseudo code implementation of this simple version of the Viterbi algorithm is given in Algorithm 4.1.

4.3.2 Backtracing the Viterbi Path

In order to reconstruct the sequence of action labels that build the best scoring path, for each tuple (t,c) the best predecessor class needs to be stored. Therefore, a traceback array $B(t,c)$ is used. In the case that no new segment starts at time t, the best predecessor class

Algorithm 4.1 Viterbi Decoding

```
1: $Q(0, :) = 1$
2: for $t = 1, \dots, T$ do
3:   for $c \in \mathcal{C}$ do
4:     $Q(t, c) = Q(t-1, c) \cdot p(x_t|c)$
5:     $B(t, c) = c$
6:     for $c' = 1, \dots, |\mathcal{C}|$ do
7:       if $Q(t, c) < Q(t-1, c') \cdot p(x_t|c) \cdot p(c|c')$ then
8:         $Q(t, c) = Q(t-1, c') \cdot p(x_t|c) \cdot p(c|c')$
9:         $B(t, c) = c'$
10: return $Q, B$
```

Algorithm 4.2 Backtracing the Viterbi Path

```
1: path := [ ]
2: $c := \arg\max_c \; Q(T, c)$
3: for $t = T, \dots, 1$ do
4:   path.append($c$)
5:   $c := B(t, c)$
6: return path.revert()
```

of the tuple (t, c) is simply the same class, i.e.

$$B(t, c) = c. \tag{4.18}$$

When looking at segment start hypothesis, the preceding class needs to be stored in $B(t, c)$. Recall that in order to define the Q function, we replaced the arg max from Equation (4.7) with the max. In order to reconstruct the best sequence, the maximizing arguments are of interest again, so the arg max needs to be used again,

$$B(t, c) = \arg\max_{c'} \left\{ Q(t-1, c') \cdot p(x_t|c) \cdot p(c|c') \right\}. \tag{4.19}$$

Starting from the best ending class at the end of the sequence, $\arg\max_c Q(T, c)$, the traceback array can be used to reconstruct the overall best path by traversing $B(t, c)$ in reverse order from T backwards following Algorithm 4.2. The traceback process is illustrated in Figure 4.3.

4.3.3 Complexity

This simple version of the Viterbi decoding allows to decode the best segmentation by means of Equation (4.7) in linear time in the number of frames. In order to obtain the best path, $Q(t, c)$ has to be computed for all frames and for all possible classes at each frame. Moreover, every value for $Q(t, c)$ can possibly be the start of a new segment, so the maximization over all predecessor classes in Equation (4.17) has to be carried out for each (t, c) tuple. Overall,

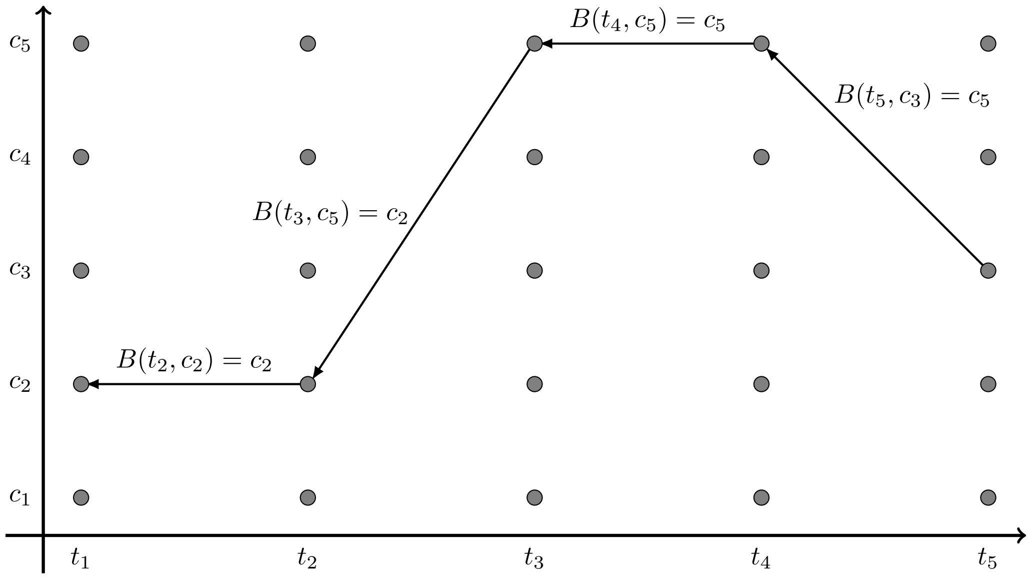


Figure 4.3: Traceback of the Viterbi path. Starting at the best class at the last frame (here: c_3), the traceback array $B(t, c)$ points to the best predecessor class at the preceding frame. The resulting segmentation in this example is $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$ with $N = 3$, $\mathbf{c}_1^N = (c_2, c_5, c_3)$ and $\mathbf{l}_1^N = (2, 2, 1)$.

the complexity of the Viterbi decoding is $\mathcal{O}(T \cdot |\mathcal{C}|^2)$. Backtracing the Viterbi path requires to find the best class for the last frame and then go backwards over the frames once and look up the respective entries in the traceback array B, see Algorithm 4.2. The complexity is $\mathcal{O}(|\mathcal{C}| \cdot T)$.

Chapter 5

Fully Supervised Action Segmentation

Building upon the general framework introduced in the previous chapter, we propose a method for temporal action segmentation given a frame-level annotation of the training videos. The approach provides concrete instantiations of the visual model, the length model, and the context model. While sophisticated methods from natural language processing are employed for the context model, the visual model is based upon classical action recognition methods that operate on a segment level, i.e. that are typically trained on pre-segmented video clips containing a single action instance and no further background frames. We analyze the proposed system and its components on several datasets and show state-of-the-art performance on Thumos 14.

Contents

5.1 Introduction

Comparing the task of classical action recognition on pre-segmented clips and temporal action segmentation on untrimmed videos, there is a huge gap in terms of performance. The advances in the classification of pre-segmented clips made in the last years do not easily transfer to action segmentation due to the difficulty of modeling action boundaries and temporal context, cf. Chapter 1 and Figure 1.1.

One crucial issue in untrimmed videos is that an action class can be arbitrarily long, e.g. a background class, but it can also be only a few frames long in a video. Without a pre-segmentation of the video, the duration of the action classes needs to be well modeled. Moreover, while video clips can already be well recognized by using only spatial information such as appearance and presence of objects (Jain et al., 2015), the differences between the frames where an activity occurs and background frames are more subtle.

The most successful methods for temporal action detection on a dataset like Thumos follow a two step approach. They first extract segments from the video using a sliding temporal window and classify each segment in a second step. The final segmentation is then achieved by greedily selecting the segments with the highest scores (Oneata et al., 2014; Rohrbach et al., 2012). More recent approaches combine these two steps into a single network such as the Faster-RCNN variant for action segmentation proposed by Chao et al. (2018). Still, these approaches perform a greedy segment selection and lack a sophisticated length and context modeling.

In this chapter, we present an approach to temporal action detection that avoids a greedy approximation and aims to find the globally most likely action sequence in a single step by solving the segmentation and classification task jointly using a variant of the Viterbi algorithm introduced in the previous chapter. The proposed model incorporates information about the duration of an action class by a length model and the temporal context by a context model. The length and context model are combined with a discriminative model for recognizing actions. The latter can be any typical action recognition system. We focus on a traditional approach using a linear classifier on top of Fisher vectors of improved dense trajectories as well as on a model based on framewise I3D features.

In the experiments, we provide an extensive analysis of our approach on three datasets where we evaluate the impact of the length and context model in detail. Our model achieves state-of-the-art accuracy for temporal action detection on the challenging Thumos benchmark.

5.2 Temporal Action Detection

We use the general model proposed in Chapter 4 with explicit assumptions for all three components, the length model, the context model, and the visual model. In the following, we describe the single components of the model. Then, the Viterbi algorithm from Chapter 4 is modified such that it can be applied to the previously defined models.

Consider the model defined in Equation (4.2),

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) \cdot p(\mathbf{l}_1^N | \mathbf{c}_1^N) \cdot p(\mathbf{c}_1^N) \Big\}. \tag{5.1}$$

The visual model, providing the actual probability of a feature sequence $\mathbf{x}_1^T$ being generated by the given segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$, is a segment-wise classifier here, so that in principle any action recognition model could be applied.

The length model, $p(\mathbf{l}_1^N | \mathbf{c}_1^N)$, determines the length of the segments. Recall that our model does not allow gaps, i.e. a segment ending at a frame t requires the next segment to start at frame $t+1$. This is no restriction if background is also modeled as an action class. The context model gets special attention here. We rely on concepts developed in natural language processing and apply smoothed m-grams to model likelihoods of action sequences. The models are described in detail in the following.

5.2.1 Context Model

We start with a detailed description of the context model. Sequences of actions $\mathbf{c}_1^N$ can be of arbitrary length. Exact inference would require to model the probability of the complete sequence $p(\mathbf{c}_1^N)$ either directly or as a product

$$p(\mathbf{c}_1^N) = \prod_{n=1}^{N} p(c_n|\mathbf{c}_1^{n-1}), \tag{5.2}$$

where the class c_n at each position n depends on all preceding classes.

Based on findings from statistical language modeling, it shows that the most relevant words in natural language sentences are those that immediately precede the current word. We transfer these observations to action classes and assume that an action class is dependent on a history of only m preceding classes,

$$p(\mathbf{c}_1^N) = \prod_{n=1}^{N} p(c_n|\mathbf{c}_1^{n-1}) \tag{5.3}$$

$$= \prod_{n=1}^{N} p(c_n|\mathbf{c}_{n-m}^{n-1}). \tag{5.4}$$

At the beginning of a sequence, the preceding classes are assumed to be virtual sequence start classes, i.e. $c_k = c_\$$ for $k \leq 0$. This kind of model is known as an m-gram (language) model.

The probabilities for each $p(c_n|\mathbf{c}_{n-m}^{n-1})$ can be estimated from the training data using the maximum likelihood method and the result are relative frequencies over each action class c for each history of m preceding classes:

$$p(c|h) = \frac{N(h,c)}{N(h,\cdot)}, \tag{5.5}$$

where h is a sequence of m preceding classes, e.g. $h = \mathbf{c}_{n-m}^{n-1}$ for $c = c_n$, and $N(h,c)$ is the count of occurrences of class c with the history h in the training data.

Note that particularly for larger histories, $N(h,c)$ may be zero and such a sequence of action classes could never be detected. Common practice is to apply smoothing to the m-grams in order to shift probability mass to unseen action sequences. In order to deal with these unseen events, we use linear discounting with backing-off (Ney et al., 1994), i.e.

$$p(c|h) = \begin{cases} (1-\lambda)\cdot\frac{N(h,c)}{N(h,\cdot)}, & \text{if } N(h,c) > 0, \\ \lambda \cdot \frac{p(c|h')}{\sum_{c':N(h,c')=0} p(c'|h')}, & \text{otherwise.} \end{cases} \tag{5.6}$$

The parameter λ assigns a certain amount of probability mass to unseen events and is obtained using maximum likelihood estimation in combination with leaving-one-out. Consider

$$N_r(h,\cdot) = |\{c : N(h,c) = r\}|, \tag{5.7}$$

i.e. N_r is a counter that returns the number of action classes c that occur exactly r times with history h. For $r = 1$, $N_r(h,\cdot)$ denotes the number of *singletons*, i.e. action classes that occur in combination with the history exactly one time. With $N_1(\cdot,\cdot)$, we denote the total number of singletons accumulated over all possible histories and N_{total} is the total number of action instances in the training data. The resulting maximum likelihood estimate for λ is then

$$\lambda = \frac{N_1(\cdot,\cdot)}{N_{\text{total}}}, \tag{5.8}$$

see Ney et al. (1994) for details.

For unseen events, the back-off model $p(c|h')$ is used in Equation (5.6). It is an m-gram of lower order, e.g. a bigram ($m = 1$) if $p(c|h)$ is a trigram ($m = 2$). Since we have a closed vocabulary, meaning that during test time no new classes occur, the lowest required back-off model is a unigram ($m = 0$) that simply models the class prior $p(c)$ without any context. The renormalization in Equation (5.6) is required for a proper probability distribution.

5.2.2 Length Model

For the length model, we assume that the length of a segment only depends on its action class but not on the preceding segment lengths or on the classes of any preceding segments. The length model $p(\mathbf{l}_1^N|\mathbf{c}_1^N)$ can then be simplified to

$$\begin{aligned} p(\mathbf{l}_1^N|\mathbf{c}_1^N) &= \prod_{n=1}^{N} p(\ell_n|\mathbf{c}_1^N, \mathbf{l}_1^{n-1}) && (5.9)\\ &= \prod_{n=1}^{N} p(\ell_n|c_n). && (5.10)\end{aligned}$$

The distribution $p(\ell|c)$ can be modeled with any discrete probability distribution defined on the natural numbers. We investigate a class-dependent and class-independent Poisson distribution as well as a class-independent length model based on the average length of all actions.

For the class-independent Poisson model, the distribution is

$$p(\ell|c) = \frac{\lambda^\ell}{\ell!} e^{-\lambda}, \tag{5.11}$$

where λ needs to be estimated from the training data. The optimal λ in the maximum likelihood sense is the average length of all actions. In case of a class-dependent Poisson model, there is a distinct parameter λ_c for each action class,

$$p(\ell|c) = \frac{\lambda_c^\ell}{\ell!} e^{-\lambda_c}, \tag{5.12}$$

and the ideal estimator is the one that assigns the mean length of action class c to λ_c.

For the third kind of length model, we define

$$p(\ell|c) = \begin{cases} 1, & \text{if } \ell \leq \mu, \\ \alpha^{\ell-\mu}, & \text{otherwise,} \end{cases} \tag{5.13}$$

where α is a decay factor and μ is the mean length all action segments occurring in the training data. In the following, this model is referred to as *mean length model*. Note that the model is not a proper probability distribution and only penalizes lengths that are larger than μ. While the Poisson models favor segments with lengths that are more likely according to the training data, this third length model version is a much less restrictive model that mainly has the purpose to avoid unreasonably long segments. Without any restriction of the length, the system would tend to hypothesize a small number of long segments in order to avoid the penalty induced by the context model each time a new segment is hypothesized.

5.2.3 Visual Model

In action classification, discriminative models such as support vector machines or convolutional neural networks achieve good performance (Wang and Schmid, 2013; Simonyan and Zisserman, 2014; Karpathy et al., 2014; Carreira and Zisserman, 2017). These kinds of models can be viewed as a class posterior distribution $p(c|\mathbf{x}_1^T)$, or $p(c|\mathbf{x}_{t_1}^{t_2})$ for action segments ranging from t_1 to t_2 in the domain of temporal action detection, respectively. Particularly when using Fisher vectors of improved dense trajectories, a linear classifier such as a support vector machine performs well (Wang and Schmid, 2013), but also an averaging of deep features such as I3D is a commonly used strategy (Carreira and Zisserman, 2017). We stick to this finding and use a log-linear model of the form

$$p(c|\mathbf{x}_{t_1}^{t_2}) = \text{softmax}(\mathbf{W}f(\mathbf{x}_{t_1}^{t_2}) + \mathbf{b}) \tag{5.14}$$

where $\mathbf{W}$ is a weight matrix, $\mathbf{b}$ the bias. For a detailed analysis of the method, we stick to improved dense trajectories as input features and denote the Fisher vector composed of these features on the video segment $[t_1, t_2]$ as $f(\mathbf{x}_{t_1}^{t_2})$. The log-linear model is also a linear classifier but in contrast to support vector machines, it directly models a class posterior distribution. The parameters $\mathbf{W}$ and $\mathbf{b}$ can be estimated from the pre-segmented training data using gradient based optimization.

In the following, we show how to incorporate such a segment-based posterior distribution into our visual model. We start with a simple factorization of $p(\mathbf{x}_1^T|\mathbf{c}_1^N, \mathbf{l}_1^N)$.

Assuming that a video segment $\mathbf{x}_{t_{n-1}+1}^{t_n}$ depends on the given segmentation $(\mathbf{c}_1^N, \mathbf{l}_1^N)$ but not on the features of the neighboring segments, we can factorize the visual model as a product of action segments,

$$p(\mathbf{x}_1^T|\mathbf{c}_1^N, \mathbf{l}_1^N) = \prod_{n=1}^{N} p(\mathbf{x}_{t_{n-1}+1}^{t_n}|\mathbf{c}_1^N, \mathbf{l}_1^N), \tag{5.15}$$

where a temporal position t_n is defined as the ending time of segment n,

$$t_n = \sum_{i=1}^{n} \ell_n. \tag{5.16}$$

Further assuming conditional independence of the video frames and assuming that a frame only depends on its class and the length of the segment it is located in, Equation (5.15) further simplifies to

$$\begin{aligned} p(\mathbf{x}_1^T|\mathbf{c}_1^N, \mathbf{l}_1^N) &= \prod_{n=1}^{N} p(\mathbf{x}_{t_{n-1}+1}^{t_n}|\mathbf{c}_1^N, \mathbf{l}_1^N) & (5.17)\\ &= \prod_{n=1}^{N} \prod_{t=t_{n-1}+1}^{t_n} p(x_t|c_n, \ell_n). & (5.18) \end{aligned}$$

Using Bayes' Theorem, Equation (5.18) can be transformed to contain a class posterior distribution,

$$p(\mathbf{x}_1^T|\mathbf{c}_1^N, \mathbf{l}_1^N) = \prod_{n=1}^{N} \prod_{t=t_{n-1}+1}^{t_n} p(c_n|x_t, \ell_n) \frac{p(x_t|\ell_n)}{p(c_n|\ell_n)}. \tag{5.19}$$

Due to the dependence on the segment length, we assume that the class posterior has the same probability for each frame within the segment, i.e.

$$\prod_{t=t_{n-1}+1}^{t_n} p(c_n|x_t, \ell_n) = p(c_n|\mathbf{x}_{t_{n-1}+1}^{t_n})^{\ell_n}. \tag{5.20}$$

Coming back to Equation (5.19), we further assume that neither the frame prior nor the class prior depend on the segment lengths. Together with Equation (5.20), this leads to

$$p(\mathbf{x}_1^T|\mathbf{c}_1^N, \mathbf{l}_1^N) = \prod_{n=1}^{N} \Big(\frac{p(c_n|\mathbf{x}_{t_{n-1}+1}^{t_n})}{p(c_n)}\Big)^{\ell_n} \prod_{t=1}^{T} p(x_t). \tag{5.21}$$

In practice, we found that a uniform class prior $p(c)$ works well, so that factor can be omitted. Moreover, the product over the frame priors $p(x_t)$ is independent of the arguments we maximize over and can thus also be omitted in the maximization.

5.2.4 Inference

Given explicit representations for the length model, the context model, and the visual model, the combined model can be formulated by inserting all these results into Equation (5.1),

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ \prod_{n=1}^{N} p(c_n|\mathbf{c}_{n-m}^{n-1}) \cdot p(\ell_n|c_n) \cdot p(c_n|\mathbf{x}_{t_{n-1}+1}^{t_n})^{\ell_n} \Big\}. \tag{5.22}$$

In order to efficiently solve the maximization problem from Equation (5.22), we use a modification of the Viterbi algorithm presented in Chapter 4. For simplicity, we derive the recursion equations for a bigram context model ($m = 1$) only. The modifications that are required for higher order context models are straightforward.

In order to enable dynamic programming over the time frames $1, \dots, T$, we transform the product from Equation (5.22) to run over the time rather than over the unknown number of segments. To simplify notation, recall the definition of $n(t)$ from Equation (4.5), a function that returns the index of a segment frame t belongs to,

$$n(t) = k \quad \text{if and only if} \quad \sum_{n=1}^{k-1} \ell_n < t \leq \sum_{n=1}^{k} \ell_n. \tag{5.23}$$

Using this notation and focusing on the maximization of Equation (5.22) for now, it can be rewritten as

$$\begin{aligned} &\max_{N,\mathbf{c}_1^N,\mathbf{l}_1^N} \Big\{ \prod_{n=1}^{N} p(c_n|\mathbf{c}_{n-m}^{n-1}) \cdot p(\ell_n|c_n) \cdot p(c_n|\mathbf{x}_{t_{n-1}+1}^{t_n})^{\ell_n} \Big\} & (5.24)\\ = &\max_{N,\mathbf{c}_1^N,\mathbf{l}_1^N} \Big\{ \prod_{t=1}^{T} \big[p(c_{n(t)}|c_{n(t)-1}) \cdot p(\ell_{n(t)}|c_{n(t)}) \cdot p(c_{n(t)}|\mathbf{x}_{t_{n(t)-1}+1}^{t_{n(t)}})^{\ell_{n(t)}}\big]^{[\![t==t_{n(t)}]\!]} \Big\}, & (5.25)\end{aligned}$$

where $[\![\cdot]\!]$ is an index function that returns one if its condition is true and zero otherwise. In this case, $t == t_{n(t)}$ is true if t is the ending time of segment $n(t)$. This way, the N factors from Equation (5.22) are sustained and the factors for the times t that are not a segment end time are one.

In order to be able to apply dynamic programming over Equation (5.25), we define an auxiliary function $Q(\tau, c)$ that specifies the best segmentation of the video up to time τ into n segments with the last segment being of class c and ending at frame τ,

$$Q(\tau, c) = \max_{\substack{n,\mathbf{c}_1^n,\mathbf{l}_1^n: \\ c_n=c, \\ \sum_{i=1}^n \ell_i=\tau}} \Big\{ \prod_{t=1}^{\tau} \big[p(c_{n(t)}|c_{n(t)-1}) \cdot p(\ell_{n(t)}|c_{n(t)}) \cdot p(c_{n(t)}|\mathbf{x}_{t_{n(t)-1}+1}^{t_{n(t)}})^{\ell_{n(t)}}\big]^{[\![t==t_{n(t)}]\!]} \Big\}. \tag{5.26}$$

Note that we bind the equation to the conditions $c_n = c$ (last segment is of class c) and $\sum_{i=1}^n \ell_i = \tau$ (accumulated length is the number of frames, τ). In a next step, the factors of the last segment can be isolated from the equation,

$$\begin{aligned} Q(\tau, c) = \max_{\substack{n,\mathbf{c}_1^n,\mathbf{l}_1^n: \\ c_n=c, \\ \sum_{i=1}^n \ell_i=\tau}} &\Big\{ p(c_n|c_{n-1}) \cdot p(\ell_n|c_n) \cdot p(c_n|\mathbf{x}_{\tau-\ell_n+1}^{\tau})^{\ell_n} \cdot \\ &\prod_{t=1}^{\tau-\ell_n} \big[p(c_{n(t)}|c_{n(t)-1}) \cdot p(\ell_{n(t)}|c_{n(t)}) \cdot p(c_{n(t)}|\mathbf{x}_{t_{n(t)-1}+1}^{t_{n(t)}})^{\ell_{n(t)}}\big]^{[\![t==t_{n(t)}]\!]} \Big\}. \end{aligned} \tag{5.27}$$

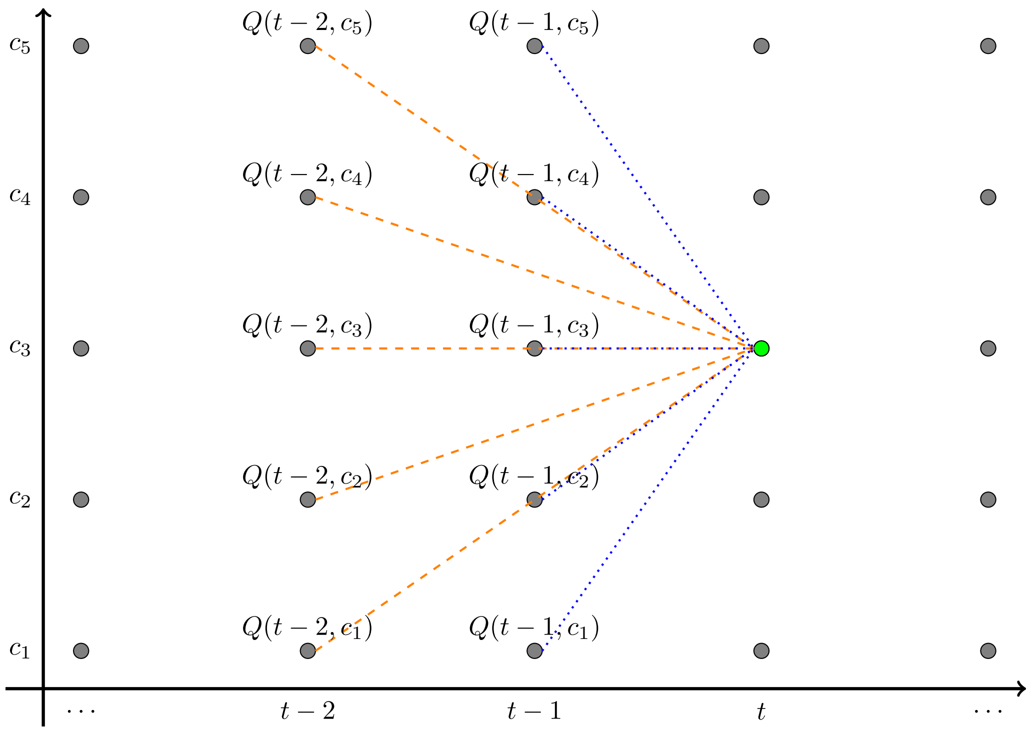


Figure 5.1: Illustration of the computation of the recursive equation $Q(\tau, c)$. For the computation of the score of a segment ending at time t in class c_3 (green dot), all possible preceding classes and all possible segment lengths have to be considered. Thus, the optimization is not only over previous segment end points $t-1$ but also over $t-2$, $t-3$, etc.

Defining $c_{n-1} = \tilde{c}$ and $\ell_n = \ell$, we can pull the isolated factors out of the maximization and rewrite

$$Q(\tau, c) = \max_{\ell, \tilde{c}} \Big[p(c|\tilde{c}) \cdot p(\ell|c) \cdot p(c|\mathbf{x}_{\tau-\ell+1}^{\tau})^{\ell} \cdot \\ \max_{\substack{n-1, \mathbf{c}_1^{n-1}, \mathbf{l}_1^{n-1}: \\ c_{n-1}=\tilde{c}, \\ \sum_{i=1}^{n-1} \ell_i = \tau-\ell}} \Big\{ \prod_{t=1}^{\tau-\ell} \big[p(c_{n(t)}|c_{n(t)-1}) \cdot p(\ell_{n(t)}|c_{n(t)}) \cdot p(c_{n(t)}|\mathbf{x}_{t_{n(t)-1}+1}^{t_{n(t)}})^{\ell_{n(t)}} \big]^{[\![t==t_{n(t)}]\!]} \Big\} \Big]. \tag{5.28}$$

Inspection of the second maximization reveals that it is $Q(\tau-\ell, \tilde{c})$, which ends our derivation of the recursive equation for $Q(\tau, c)$:

$$Q(\tau, c) = \max_{\ell, \tilde{c}} \Big\{ p(c|\tilde{c}) \cdot p(\ell|c) \cdot p(c|\mathbf{x}_{\tau-\ell+1}^{\tau})^{\ell} \cdot Q(\tau-\ell, \tilde{c}) \Big\}. \tag{5.29}$$

A visual illustration of the decoding process is shown in Figure 5.1. In order to compute the score of class c_3 ending at time t (green dot), the hypothesis of all possible preceding classes and all possible preceding lengths have to be looked at. If the preceding segment is assumed

Algorithm 5.1 Viterbi Decoding

```
1: $Q(0,:) = 1$
2: $Q(1:T,:) = 0$
3: for $t = 1, \dots, T$ do
4:   for $c \in \mathcal{C}$ do
5:     for $c' \in \mathcal{C}$ do
6:       for $l = 1, \dots, \min(t, L)$ do
7:         if $Q(t,c) < p(c|\tilde{c}) \cdot p(\ell|c) \cdot p(c|\mathbf{x}_{t-\ell+1}^{t})^{\ell} \cdot Q(t-\ell, \tilde{c})$ then
8:           $Q(t,c) = p(c|\tilde{c}) \cdot p(\ell|c) \cdot p(c|\mathbf{x}_{t-\ell+1}^{t})^{\ell} \cdot Q(t-\ell, \tilde{c})$
9:           $A(t,c) = l$
10:          $B(t,c) = c'$
11: return $Q, A, B$
```

to be ending at $t-1$ (blue lines), the current segment has length one and the quantity being computed for the arc from $Q(t-1,\tilde{c})$ to $Q(t,c_3)$ is $p(c_3|\tilde{c}) \cdot p(\ell = 1|c_3) \cdot p(c_3|\mathbf{x}_t^t)^1 \cdot Q(t-1,\tilde{c})$. If the preceding segment is assumed to be ending at $t-2$ (orange lines), the current segment has length two and the quantity being computed for the arc from $Q(t-2,\tilde{c})$ to $Q(t,c_3)$ is $p(c_3|\tilde{c}) \cdot p(\ell = 2|c_3) \cdot p(c_3|\mathbf{x}_{t-1}^t)^2 \cdot Q(t-2,\tilde{c})$. For every (τ, c) tuple, this involves running over all possible preceding classes and all possible segment lengths. A pseudo code implementation of the modified Viterbi decoding is provided in Algorithm 5.1. While Equation (5.29) originally requires a maximization over all possible action lengths at each frame τ, this can be prohibitive for long videos in practice as it leads to quadratic runtime in the number of frames T, considering that ℓ has to run from 1 to τ at frame τ. We therefore introduce a constant L that limits the maximal length of an action. In practice, this is reasonable as the duration of an action inherently depends on the action itself and not on the overall duration of the video. The score of the best segmentation in the sense of Equation (5.22) is now given by $\max_c Q(T,c)$. In order to reconstruct the best action segmentation, two additional traceback arrays need to be stored:

$$A(\tau, c) = \arg\max_{\ell} \Big\{ \max_{\tilde{c}} \; \big\{ p(c|\tilde{c}) \cdot p(\ell|c) \cdot p(c|\mathbf{x}_{\tau-\ell+1}^{\tau})^{\ell} \cdot Q(\tau-\ell, \tilde{c}) \big\} \Big\} \tag{5.30}$$

is the best-scoring length of the segment with class c ending at time τ and

$$B(\tau, c) = \arg\max_{\tilde{c}} \Big\{ \max_{\ell} \; \big\{ p(c|\tilde{c}) \cdot p(\ell|c) \cdot p(c|\mathbf{x}_{\tau-\ell+1}^{\tau})^{\ell} \cdot Q(\tau-\ell, \tilde{c}) \big\} \Big\} \tag{5.31}$$

is the best predecessor class of the segment ranging from $\tau - A(\tau, c) + 1$ to τ that is hypothesized with class c. The optimal segmentation can then be reconstructed using Algorithm 5.2. Starting at the last frame T and the best ending class c at this frame, the best hypothesized segment length is $A(T,c)$ and the best class of the predecessor segment is $B(T,c)$. In the next step, the best ending segment at time $T - A(T,c)$ is backtraced. This way, for a segmentation with N segments, the algorithm returns a list of N label/length pairs that define the segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$.

Algorithm 5.2 Reconstruction of the best segmentation

```
1: segments = [ ]
2: $t = T$
3: $c = \arg\max_{\tilde{c}} Q(T, \tilde{c})$
4: while $t > 0$ do
5:     $\ell = A(t, c)$
6:     $c = B(t, c)$
7:     $t = t - \ell$
8:     segments.append$\big((c, \ell)\big)$
9: return segments.revert()
```

Complexity

Since for each frame each possible action length needs to be evaluated, and for each class each predecessor class has to be considered, inference is quadratic in the number of frames and classes, i.e. $\mathcal{O}(|\mathcal{C}|^2T^2)$. With the upper limit L on allowed action lengths, however, it can easily be reduced to $\mathcal{O}(|\mathcal{C}|^2LT)$, see Algorithm 5.1. For higher order context models, predecessor classes also need to be stored. For a trigram context model, for instance, a function $Q(\tau, c, \tilde{c})$ needs to be computed, which increases the runtime to $\mathcal{O}(|\mathcal{C}|^3LT)$. In practice, however, the runtime is dominated by the computation of the visual model, which is not affected by an increased history in the context model. Thus, the difference of using a bigram or a trigram context model only has a minor impact on the overall runtime.

The runtime of the traceback is – similar to the simpler case presented in Chapter 4 – still $\mathcal{O}(T)$. Inspection of Algorithm 5.2, however, reveals that there are at most N traceback steps and the worst case runtime only appears if $N = T$. While in principle $N = T$ is possible, i.e. in the best segmentation, each frame is its own segment, in practice, usually $N \ll T$. Note that in contrast to the previously presented, simpler case in Algorithm 4.2, here we return a segmentation directly rather than a framewise labeling.

5.3 Experiments

We provide an extensive evaluation of the framework for fully supervised temporal action segmentation. In a first step, we discuss the context model, evaluate how the model components work together, and provide a discussion on the choice of different length models for different datasets. We then provide a comparison of our approach to early sliding window based approaches using traditional, hand-crafted features. Eventually, we use recent deep I3D features in combination with the proposed framework and compare our work to other state-of-the-art approaches that are based on deep neural networks.

5.3.1 Setup

We briefly summarize the setup of our experiments for the in-depth analysis. A detailed analysis of the proposed approach is provided on Thumos. We use the 200 temporally annotated videos of the validation set to train the context model, length model, and visual model. For testing, the 212 videos of the official test set are used. In order to compare our method to early approaches using sliding windows, we also report results on MPII Cooking and 50 Salads. Details on the datasets are outlined in Chapter 2.

The visual model is trained by segmenting the training data according to the ground truth and computing a Fisher vector of improved dense trajectories as described in Wang and Schmid (2013) for each segment. The ground truth annotation of the training data is also used to estimate the length- and context model.

For detection, we extract unnormalized Fisher vectors of improved dense trajectories for each video frame and store the result as a sequence of cumulatively summed vectors. Summing over the temporal dimension allows to efficiently compute the averaged input features. Particularly, let $\mathbf{x}_1^T$ be the input sequence of framewise unnormalized Fisher vectors and $\hat{\mathbf{x}}_1^T$ be the result of summing $\mathbf{x}_1^T$ over the temporal domain, i.e.

$$\hat{x}_t = \sum_{\tau=1}^{t} x_\tau \qquad \text{for each } t \in \{1, \dots, T\}. \tag{5.32}$$

The Fisher vector for an arbitrary hypothesized segment can now be computed efficiently by looking up the segment start and end time in $\hat{\mathbf{x}}_1^T$ and applying power- and ℓ_2-normalization. The final Fisher vector representation of $\mathbf{x}_{t_1}^{t_2}$, for instance, is then given by

$$FV(\mathbf{x}_{t_1}^{t_2}) = \ell_2\text{-norm}\Big(\text{powernorm}\big(\frac{\hat{x}_{t_2} - \hat{x}_{t_1-1}}{t_2 - t_1 + 1}\big)\Big). \tag{5.33}$$

We compare our method to a sliding window baseline similar to the one used in Rohrbach et al. (2012). Starting with a window size of 30 frames and a step size of 10 frames, both values are increased by a factor of $\sqrt{2}$ until the window size exceeds $1,000$ frames. Non-maximum suppression is then applied to remove all overlapping windows. As classifier, we use the log-linear model from Equation (5.14) that is also used in our method.

For our approach, the maximal action length is limited to $1,000$ frames and we increment τ by 10 instead of 1 for the computation of $Q(\tau, c)$ in Equation (5.29), i.e. we only evaluate every 10th frame to reduce runtime. If not mentioned otherwise, we use the mean length model from Equation (5.13). With these settings, our algorithm needs 7.5h on a CPU with eight 1.2GHz cores for inference on Thumos.

For the evaluation on Thumos, we use the official evaluation script provided by Idrees et al. (2017). The script computes the mean average precision (mAP) over the detections. A detection is marked as correct if its intersection over union ratio is larger than some overlap threshold, cf. Section 3.2. Since we find this evaluation method very useful as it also gives insight in how a method performs for various overlap ratios, we also apply it to MPII Cooking and 50 Salads. For MPII Cooking, we additionally report precision and recall as well as single mAP based on the midpoint hit criterion as proposed in Rohrbach et al. (2012) to be able to compare to other methods using this dataset.

		Perplexity	
m-gram		validation	test
unigram	$(m = 0)$	7.999	8.401
bigram	$(m = 1)$	3.484	4.204
trigram	$(m = 2)$	1.176	1.203

Table 5.1: Perplexities of different m-gram context models on the Thumos validation set (used for training) and test set (unseen data).

		Overlap				
m-gram		0.1	0.2	0.3	0.4	0.5
no context model		28.5	26.6	22.2	17.0	10.6
unigram	$(m = 0)$	19.5	17.8	15.0	11.8	8.6
bigram	$(m = 1)$	36.6	33.4	27.7	21.9	15.2
trigram	$(m = 2)$	39.7	35.7	30.0	23.2	15.2

Table 5.2: Effect of the context model order on Thumos. Results are reported as mAP for different overlap ratios.

5.3.2 Evaluation of the Context Model

In this section, we evaluate the effect of the context model on the performance of the system. To this end, we trained different kinds of context models on the temporally annotated validation set and compared their strength and their effect on the detection.

The strength of a context model can be measured using the perplexity (Bahl et al., 1983; Ney et al., 1994). For a single sequence with N action classes, it is defined as

$$\text{PP} = \Big(\prod_{n=1}^{N} p(c_n | \mathbf{c}_{n-m}^{n-1}) \Big)^{-\frac{1}{N}}. \tag{5.34}$$

The perplexity for a dataset consisting of multiple sequences is the product of the context model probabilities for each sequence where N is replaced by the total number of segments in the dataset. Intuitively, the perplexity can be seen as the number of possible choices per position. A small perplexity corresponds to a strong context model.

Table 5.1 shows the perplexities for different m-gram context models on Thumos. Note that the perplexities on the training data (i.e. the validation set) and on the test data are very similar, indicating that the learned context model works well for the action context on the test set. Further, the perplexity decreases with increasing m-gram order, making the context model stronger.

Results of the complete system using different context models are shown in Table 5.2. The system with the unigram context model (no history, i.e. $m = 0$) performs clearly worse than the system without a context model. Since 50% of the classes in Thumos are background,

		avg. length	Overlap				
			0.1	0.2	0.3	0.4	0.5
(a)	complete system (Eq. (5.22))	182.1	39.7	35.7	30.0	23.2	15.2
Visual Model							
(b)	visual model with class prior	108.0	31.0	29.5	18.7	11.3	6.6
(c)	visual model w/o length factor	117.3	33.4	29.8	23.7	16.6	10.8
Length- and Context Model							
(d)	w/o context model	147.4	28.5	26.6	22.2	17.0	10.6
(e)	w/o length model	422.8	20.9	16.6	12.8	8.6	4.9
(f)	w/o context and length model	543.8	13.5	11.1	8.6	6.5	4.1
(g)	w/o context, length, and length factor	643.1	10.4	8.9	6.8	5.3	3.5

Table 5.3: Effect of the model components on Thumos. In the second column, the average length of the detected action segments is given. The average length of actions in the ground truth is 212.5 frames.

the model has a strong bias towards background. Moreover, background usually gets a high classification score for any segment, so the model tends to predict multiple consecutive background segments. This can be prevented by taking context into account. A bigram ($m = 1$) already leads to a huge gain in performance. Using more context, e.g. a trigram ($m = 2$), can still boost the performance, although the gain is not as intense as for the bigram. For the remainder of the paper, we use a trigram context model as it produces the best results. Note that higher order context models are not promising due to the already extremely low perplexity of the trigram model. Even on the test set, the model only has the choice of 1.2 different action classes on average. The characteristics on Thumos make higher order models unreasonable, particularly because the dataset mainly follows the structure *actionX, background, actionX, background, ...* and a history of two action classes already ensures that the alternating pattern between an action of a specific class and background is kept.

5.3.3 Model Components

In Table 5.3, the impact of each component is analyzed. In addition to the results at each of the five overlap ratios, we also report the average length of the detected segments in frames. The detection result of a video from the Thumos test set in Figure 5.2 serves as an example for the cases discussed in the table.

We start with an analysis of the visual model. In Section 5.2.3, we argue that a uniform prior $p(c)$ in Equation (5.21) works well in practice. If we use a non-uniform class prior (Table 5.3 (b)), the performance is far below the performance of the uniform prior (Table 5.3 (a)) and the average segment length is shorter. This is due to the fact that the division by the prior emphasizes infrequent classes. Hence, longer actions are more likely to be split into

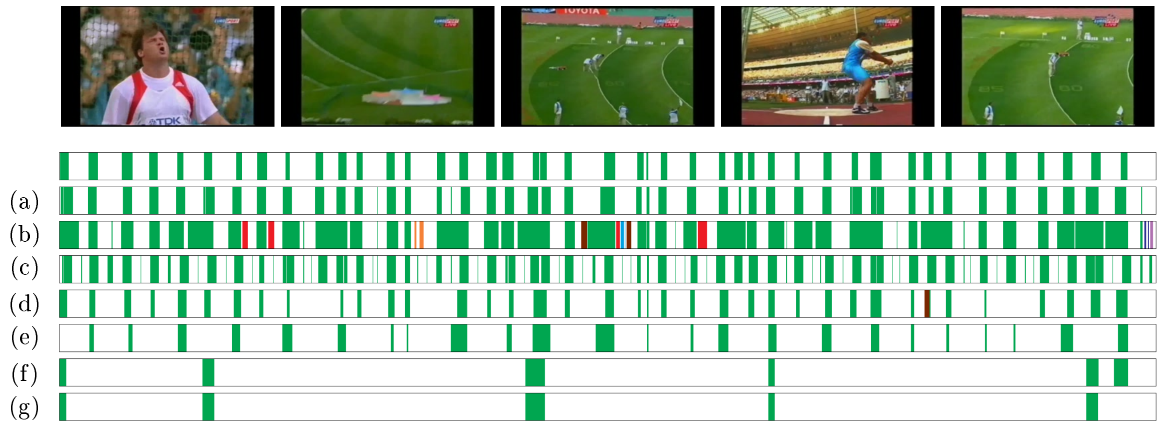


Figure 5.2: Detection result on `video_test_0001058` of Thumos. The video is 15 minutes long and contains actions of the class `Hammer Throw`. The first row contains the ground truth, the other rows show the detection results for the systems (a)-(g) from Table 5.3. Different classes have different colors.

multiple short segments of rare classes which are then sometimes falsely classified, see Figure 5.2 (b).

Due to the interplay between the action, length, and context model, the impact of the power factor is a little bit more complex. Without the length and context model, the power factor ℓ_n penalizes long segments, cf. (f) and (g) in Table 5.3. However, when length and context model are included, ℓ_n has another effect: It enhances the visual model compared to the context- and length model. So, omitting ℓ_n increases the impact of the length model. Thus, sequences that are longer than μ (see Equation (5.13)), typically background, are more likely to be split. The context model strongly penalizes consecutive background segments, which explains the short action artifacts in Figure 5.2(c).

Without length- and context model, long segments which include multiple short actions are classified as background since most of the frames are actually from the background. Thus, performance drops and the average segment length increases, cf. (f) in Table 5.3 and Figure 5.2.

When using a context model without length model, the performance is still not satisfying and the average segment length is quite large, see (e) in Table 5.3 and Figure 5.2. The reason is that each time a new segment is hypothesized, a context model probability is multiplied to the probability score of the system. Thus, there is a clear tendency towards few, long segments in order to avoid context model penalties. Adding a length model compensates for this effect since unreasonably long segments are penalized. Note the interdependence of both models. While the complete system which includes both performs well, the effect of the length model is too strong if the context model is omitted, cf. (d) in Table 5.3. The hypothesized segments are rather short in this case and the performance also drops again. Moreover, false detections occur due to the loss of context information, see Figure 5.2 (d).

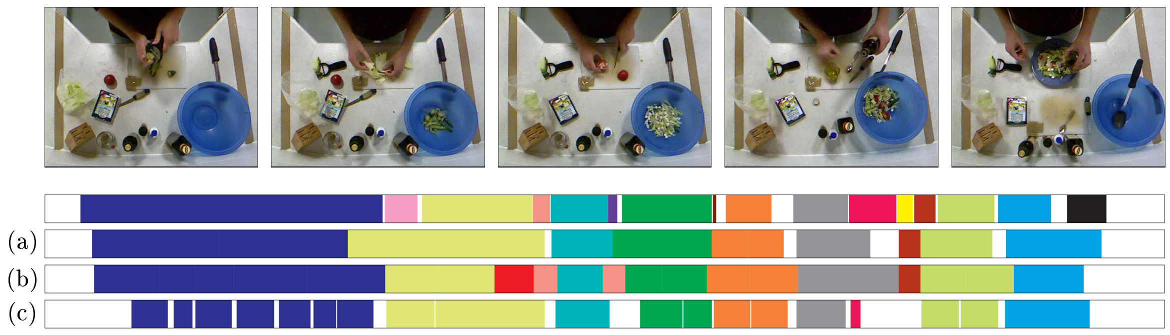


Figure 5.3: Detection result on `rgb-03-1` of 50 Salads. Each class is encoded by another color, background is white. The first row contains the ground truth, the other rows show the detection results of our system with (a) a class-dependent Poisson model, (b) a class-independent Poisson model, and (c) the mean length model.

Thumos	MPII Cooking	50 Salads
2.77	1.20	1.31

Table 5.4: χ^2-distance between the ground truth length distribution and the Poisson model averaged over all classes.

5.3.4 Length Model

We also evaluate our method on MPII Cooking and 50 Salads in addition to Thumos, starting with an investigation of three different kinds of length models. The choice of the length model depends on the dataset and the characteristics of the action classes. A strong length model, such as the class-dependent Poisson model, is superior on 50 Salads and MPII Cooking but performs worse on Thumos, see Table 5.5. To analyze this effect, we compare the distribution of the ground truth lengths of each class with the distribution generated by the Poisson model. We discretize both distributions as a histogram with 20 bins with a width of 50 frames each. Then, we compute the χ^2-distance between the ground truth distribution d_{gt} and the distribution d_{Poi} generated by the Poisson model,

$$\text{dist}_{\chi^2} = \sum_{b=1}^{B} \frac{(d_{\text{gt}}(b) - d_{\text{Poi}}(b))^2}{d_{\text{gt}}(b) + d_{\text{Poi}}(b)}, \tag{5.35}$$

where b denotes the bin index. We report the mean distance over all classes in Table 5.4. While the χ^2-distances for MPII Cooking and 50 Salads are comparably small, the value for Thumos is twice as large, indicating that the Poisson model is a worse representation of the true length distribution on Thumos than on the other datasets.

The mean length model, which only compensates for the bias of the context model towards long segments, performs best on Thumos where the Poisson distribution is a poor model of the underlying distribution as shown in Table 5.4. Even on 50 Salads and MPII Cooking

	Overlap				
Method	0.1	0.2	0.3	0.4	0.5
Thumos					
sliding window	32.5	27.9	20.6	15.0	8.6
Univ. of Florence (Karaman et al., 2014)	4.6	3.4	2.4	1.4	0.9
CHUK & SIAT (Wang et al., 2014)	18.2	17.0	14.0	11.7	8.3
INRIA (challenge winner) (Oneata et al., 2014)	36.7	33.4	27.0	20.8	14.4
ours w/ mean length model	**39.7**	**35.7**	**30.0**	**23.2**	**15.2**
ours w/ class-independent Poisson model	33.7	30.7	25.5	19.1	12.7
ours w/ class-dependent Poisson model	25.1	23.4	20.1	14.4	8.8
MPII Cooking					
sliding window	22.2	19.7	15.8	12.6	7.9
ours w/ mean length model	22.0	20.9	18.0	13.5	10.4
ours w/ class-independent Poisson model	21.9	20.0	16.3	12.9	9.8
ours w/ class-dependent Poisson model	**24.8**	**23.9**	**22.0**	**19.2**	**14.0**
50 Salads					
sliding window	20.1	15.8	12.6	10.0	8.0
ours w/ mean length model	30.5	29.5	26.0	20.8	15.3
ours w/ class-independent Poisson model	37.5	35.7	30.6	23.7	14.9
ours w/ class-dependent Poisson model	**37.9**	**36.8**	**35.2**	**31.2**	**22.9**

Table 5.5: Comparison of our method to early sliding window based approaches on the three datasets Thumos, MPII Cooking, and 50 Salads. We use the evaluation protocol proposed for Thumos for all three datasets here.

where explicit length modeling is superior, the mean length model outperforms the sliding window baseline. We also investigate the decay factor α from Equation (5.13). For values between 0.5 and 0.9, the results do not change substantially. Only if α is very close to one, the effect of the length model vanishes since long segments are not anymore penalized. In this case, the performance decreases towards the system without length model, cf. Table 5.3e.

The class-independent Poisson model is a model in between, not as strong as the class-dependent model, but more explicit than the mean length model. Only on Thumos, where the true length distribution is hard to fit, it performs better than the class-dependent model. On the other datasets, the class-dependent model is still superior.

An example detection for each of the three length models on a video from 50 Salads is illustrated in Figure 5.3. In contrast to the class-dependent model (Figure 5.3a), the class-independent Poisson model (Figure 5.3b) tends to avoid short segments, particularly for the background class. The mean length model (Figure 5.3c) prefers short segments, which results in an over-segmentation of long actions.

A lower bound on the segment lengths is defined by the subsampling of frames. On 50

Method	Multi-class prec	Multi-class recall	per class mAP
sl. window, holistic (Rohrbach et al., 2012)	17.7	40.3	44.2
+ pose features (Rohrbach et al., 2012)	19.8	40.2	45.0
multiple granularity (Ni et al., 2014)	28.6	48.2	54.3
progressive interactional object parsing (Ni et al., 2016)	34.8	**51.7**	**58.9**
ours	**45.0**	25.9	58.3

Table 5.6: Multi-class precision and recall and single class mAP on MPII Cooking. We used the evaluation protocol proposed by Rohrbach et al. (2012).

Salads, the 0.1 overlap mAP slightly decreases from 39.1% to 37.9% and 37.6% for 5, 10, and 20 frame subsampling. For efficiency reasons, we stick to the 10 frame subsampling for all experiments.

5.3.5 Comparison to Early Sliding Window Based Approaches

Our method outperforms the sliding window baseline consistently on all datasets, see Table 5.5. On Thumos, our system achieves 3% higher mAP for overlap 0.1 to 0.4 and is still 1% better for overlap 0.5 compared to the winning submission from INRIA (Oneata et al., 2014). Their approach is based on a sliding window and a model combination of their system from the classification challenge and a trajectory based classifier trained on the data for the detection task. Our system also outperforms the other challenge submissions of Karaman et al. (2014) and Wang et al. (2014) which both use a sliding window. Wang et al. (2014) additionally include CNN features in their classifier. Table 5.6 shows multi-class precision/recall and single class mAP on MPII Cooking. Rohrbach et al. (2012) use dense trajectories as features (holistic) and additional pose features. Ni et al. (2014) use dense trajectory features and detect hand-object interactions. In a follow up work Ni et al. (2016) use LSTMs to parse interactional objects that help action detection. While the existing approaches achieve a high recall at the cost of a comparably low precision, our approach achieves a higher precision at the cost of lower recall. In terms of single class mAP, we are 14% better than Rohrbach et al. (2012) and 4% better than Ni et al. (2014). The work of Ni et al. (2016) has comparable performance to our approach but a lower precision.

5.3.6 Comparison to State-of-the-Art

In this section, we evaluate our model with recent deep I3D features (cf. Section 3.7) and compare the results against the current state-of-the-art on Thumos. The used I3D features are extracted from a network trained on Kinetics, the largest available action recognition dataset so far. Most of the 20 Thumos classes are covered by the 400 classes of Kinetics and the features have generally shown great performance when used on other datasets than Kinetics, e.g. on UCF-101, see Carreira and Zisserman (2017) for details.

	segment classification accuracy (%)
I3D + linear	82.2
I3D + BoW-eq. NN	85.8

Table 5.7: Segment accuracy on pre-segmented action clips of the 212 videos from the Thumos test set. For training, pre-segmented action clips have been extracted from the 200 videos of the Thumos validation set.

We extract the features and train a classifier for the 20 classes plus background on the validation set of Thumos. Therefore, we segment the videos into clips covering exactly one action instance or a background instance. This way, we obtain $5,591$ segments that can be used to train the segment classifier. The trained segment classifier predicts class probabilities $p(c|\mathbf{x}_{t_1}^{t_2})$ for a segment ranging from t_1 to t_2. Note that this kind of model exactly matches the required distribution in the visual model, see Equation (5.21), so it can be used in the proposed action segmentation framework.

We investigate two architectures. For the first, the extracted framewise I3D features are averaged for each action segment and a linear classifier is trained on these averaged features, which corresponds to a visual model similar to the one used before and described in Section 5.2.3. The second architecture follows the BoW-equivalent neural network described in Section 3.6. Following the network architecture shown in Figure 3.6, a softmax layer is used for soft quantization, temporal averaging yields a vectorized representation of all frames of the segment, and the explicit feature map replaces the kernel a SVM would typically use. We use $4,000$ units in the first hidden softmax layer, corresponding to $4,000$ visual words, and apply the χ^2-feature map with sampling parameter $n = 2$, resulting in a $20,000$-dimensional feature vector that is fed into the final classification layer.

In Table 5.7, the segment classification accuracy is shown for the linear model and the BoW-equivalent neural network. The latter clearly outperforms the linear model and is thus also a promising candidate to be used in the action segmentation framework.

Recent works on the Thumos temporal action detection task are mainly focusing on deep end-to-end architectures. We provide an extensive comparison in Table 5.8 and run our system with I3D features as a comparison. The most relevant works are the ones by Zhao et al. (2017) and Chao et al. (2018). The first relies on segment proposals that are divided into start, main, and end parts. Temporal pyramid pooling and a classification module then lead to a segment prediction. The second work is inspired by one of the most popular architectures for object detection, Faster-RCNN (Ren et al., 2015), and models the task of temporal action detection as the 1D equivalent to object detection. These two methods show complementary results. If the overlap threshold is small, i.e. a segment is already considered correct if it only overlaps with 10% or 20% with the ground truth, the method of Zhao et al. (2017) performs best. For very tight overlap thresholds, the work of Chao et al. (2018) is the most accurate. Note that a high overlap ratio is less forgiving for methods that tend to generate an over-segmentation. Imagine a well located segment is cut in half in the middle by a few falsely assigned background labels. At overlap 0.5, both the left and right half do

	Overlap				
	0.1	0.2	0.3	0.4	0.5
State-of-the-art approaches					
Frame Glimpses (Yeung et al., 2016)	48.9	44.0	36.0	26.4	17.1
Multistage CNNs (Shou et al., 2016)	47.7	43.5	36.3	28.7	19.0
Structured Max Sums (Yuan et al., 2017)	51.0	45.2	36.5	27.8	17.8
Cascaded Boundary Regression (Gao et al., 2017b)	60.1	56.7	50.1	41.3	31.0
R-C3D (Xu et al., 2017)	54.5	51.5	44.8	35.6	28.9
BSN + UNet (Lin et al., 2018)	–	–	53.5	45.0	36.9
Structured Segment Networks (Zhao et al., 2017)	**66.0**	59.4	51.9	41.0	29.8
Re-thinking Faster-RCNN (Chao et al., 2018)	59.8	57.1	53.2	**48.5**	**42.8**
frame-level approaches (I3D features)					
frame-level linear classifier	61.2	56.6	48.0	35.1	25.1
frame-level BoW-eq. NN	65.2	60.0	52.7	42.1	29.7
full action segmentation framework (I3D features)					
with linear segment classifier as visual model	62.0	58.2	52.0	42.2	32.8
with BoW-eq. NN as visual model	63.8	**60.3**	**55.7**	47.2	35.8

Table 5.8: Comparison of recent deep learning based approaches on Thumos compared to our method with I3D features.

not have sufficient overlap with the ground truth to be considered correct. At overlap 0.1, on the contrary, one of the segments would still be counted as a correct detection.

In a first comparison, we evaluate the performance of deep I3D features against state-of-the-art results. Therefore, we used the trained segment classifier – the linear model and the BoW-equivalent neural network – to output a prediction for every single frame. In order to obtain framewise predictions, I3D features within a 10 frame window around the current frame are passed to the respective network and the class with the highest score is assigned to the frame. Note that no context model, length model, or Viterbi decoding is applied to the output. Interestingly, this very simple baseline already achieves great results. Particularly on high overlap ratios, the performance is close to the current state-of-the-art. The frame-level BoW-equivalent neural network performs on par with the work of Zhao et al. (2017), which is interesting since the approach is essentially a frame-level classifier. This indicates that recent advances on Thumos are much more attributed to strong deep features than to actual segmentation networks. Moreover, since this approach is strong for low overlap ratios but clearly worse than the approach of Chao et al. (2018) on higher overlap ratios, it is questionable if complex approaches that only perform well on low overlap ratios should be favored over simple frame-level approaches at all.

In a next step, we combine the two I3D based frame-level baselines with our framework, i.e. we use the classifiers in the visual model and apply the full pipeline including context

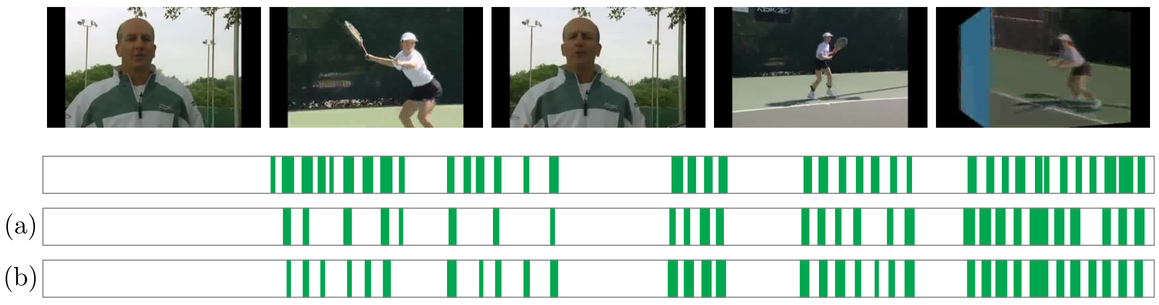


Figure 5.4: Detection result on `video_test_0000026` on Thumos. The video is 3:28 minutes long and contains actions of the class `Tennis Swing`. The top row shows the ground truth, (a) is our approach with a linear segment classifier and (b) is our approach with the BoW-equivalent neural network as segment classifier. The latter produces a better segmentation and misses less action instances.

model, length model, and Viterbi decoding. Comparing the last four rows of Table 5.8, it can be seen that incorporation of the context and length model as well as the sophisticated Viterbi decoding results in significant improvements for the higher and more difficult overlap ratios. Our results are somewhere in between the results of Zhao et al. (2017) and Chao et al. (2018), producing good segmentations at all overlap ratios and outperforming other state-of-the-art approaches at overlap ratios of 0.2 and 0.3.

Figure 5.4 shows an example segmentation of our method using (a) the linear segment classifier and (b) the BoW-equivalent neural network as segment classifier. The linear segment classifier is less reliable than the BoW-equivalent neural network and does not find the action instances as reliably.

5.4 Summary

In this chapter, we proposed a method for temporal action detection that jointly models the segmentation and recognition of actions. The approach includes a length and context model in addition to an action classifier. Using dynamic programming, we can efficiently infer the globally optimal action segmentation and classification.

An analysis of the impact of each model component revealed that the combination of length and context model is crucial for good performance. An investigation of three different length models on each of the three datasets revealed that a strong length model, e.g. a class-dependent Poisson model, is beneficial if it represents the true distribution of the action lengths. While a length model alone usually leads to over-segmentation if short segments are penalized less than long segments, the context model has the opposite effect as for each segment, a context model probability is multiplied and reduces the overall likelihood. Both models combined counteract their respective weaknesses and lead to major improvements. We have shown state-of-the-art results on multiple datasets, particularly on Thumos, where we compared to recent end-to-end architectures.

Chapter 6

RNN-HMMs for Weakly Supervised Action Segmentation

In the previous chapter, we discussed temporal action segmentation in a fully supervised setting, i.e. when the labels of each frame of the training videos are known. For practical applications, this is frequently an unreasonable setting. Obtaining full frame-level annotation is expensive and time consuming. Thus, there is a need to reduce the amount of required annotation. In this chapter, we propose a method for training an action segmentation system using less supervision in form of action transcripts. Action transcripts are a complete, ordered sequence of actions occurring in the video. In contrast to frame-level annotations, they do not contain explicit temporal information or frame-to-label correspondences. Therefore, they are much easier to obtain.

In order to effectively learn an action segmentation model, frame-to-label correspondences have to be established either implicitly or explicitly during training. We describe a training framework that allows for weakly supervised learning and is based on a modification of the general framework consisting of the visual, length, and context model that also built the foundation for the fully supervised approach.

Contents

6.1 Introduction

Hiring human annotators to generate a fully labeled video dataset is expensive and when it comes to human actions in videos, transitions are frequently fuzzy and ambiguous, so that inexperienced annotators do not always agree on temporal action boundaries. Moreover, given a larger amount of action classes, frame-level annotation is challenging as each annotator needs to know the appearance and characteristics of every relevant action class. This problem is further aggravated by the need for large scale training data for most deep learning approaches as shown by Carreira and Zisserman (2017). Additionally, the cost of training data collection makes it hard to acquire enough data to advance concepts beyond short clips, e.g. towards long term temporal models.

As pointed out in Section 1.2.2, one way to address this issue is to give up on the need of frame-based annotation and to use only action labels and their ordering information to learn the respective action classes. This information is much easier to generate for human annotators, or can even be automatically derived from scripts (Laptev et al., 2008; Marszalek et al., 2009) or subtitles (Alayrac et al., 2016). First attempts to address this kind of problem have been made by Bojanowski et al. (2014); Alayrac et al. (2016); Huang et al. (2016) and Kuehne et al. (2017).

In this context, we propose a hierarchical approach to address the problem of weakly supervised learning of human actions from transcripts. The method combines recognition in a coarse-to-fine manner. On the fine grained level, we use a discriminative representation of subactions, modeled by a recurrent neural network as e.g. used by Donahue et al. (2015); Ng et al. (2015); Singh et al. (2016), or Wu et al. (2015b). In our case, the RNN is used as a basic recognition model as it provides robust classification of small temporal chunks. This allows to capture local temporal information. The RNN is supplemented by a hidden Markov model that serves as a coarse probabilistic model to allow for temporal alignment and inference over long sequences.

To bypass the difficulty of modeling long and complex action classes, we divide all actions into smaller building blocks. Those subactions are eventually modeled within the RNN and later combined by the inference process. The usage of subactions allows to distribute heterogeneous information of one action class over many subclasses and to capture characteristics such as the length of the overall action class. We show that automatically learning the number of subactions for each action class leads to a notably improved performance.

The obvious advantage of this kind of model is that it allows recognition of fine grained movements by still capturing mid and long temporal relations between frame responses. But the model is also especially suitable for the task of weak learning because it enforces a modular structure, as frame based responses are first modeled on subaction level and then combined to action classes and eventually to action sequences. This allows for an iterative refinement of fine-grained and coarse recognition as well as an alternating adaptation of both elements.

Our model is trained with an iterative procedure. Given the weakly supervised training data, an initial segmentation is generated by uniformly distributing all actions among the video. For each action segment, all subactions are uniformly distributed among the part of the video belonging to the corresponding action. This way, an initial alignment between video frames and subactions is defined. In an iterative phase, the RNN is then trained on this alignment and used in combination with the coarse model to infer new action segment boundaries. From those boundaries, we recompute the number of subactions needed for each action class, distribute them again among the frames aligned to the respective action, and repeat the training process until convergence.

To further improve the performance in this context, we extend the standard HMM formulation by the introduction of a state length regularizer during inference. The length regularization aims to balance the temporal dynamics of the system. The intuition underlying this concept is that actions are usually not only characterized by their specific movements, but also by the duration that is necessary to execute a certain task. One way to include this characteristic in the proposed system is to model the length of an action by the number of HMM states used to represent the action. But we found that depending on the observation prior, a small number of states will aggregate all frames of an action during inference while the other states are quickly skipped. This undermines the original idea of representing variable length actions by adapting the number of states. The proposed length regularization forces the model to stay in each HMM state for a limited, reasonable time only.

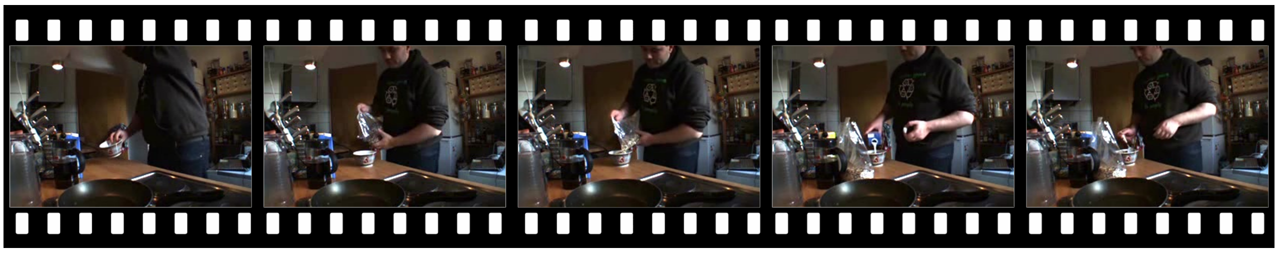
We evaluate the approach proposed in this chapter on two common benchmark datasets, the Breakfast dataset (Kuehne et al., 2014) and Hollywood Extended (Bojanowski et al., 2014) for the task of temporal action segmentation as well as for temporal action alignment. While training for both tasks is the same, recall that for the latter action transcripts are given during inference, whereas for temporal action segmentation only the video is given without any information about the occurring labels.

The remainder of this chapter is organized as follows. In Section 6.2, the setting of weakly supervised learning using action transcripts is described. In Section 6.3, the proposed system is introduced in detail, addressing the hierarchical action model formulation in general, the fine-grained subaction classification, the formulation of the length regularization, the inference over video sequences, the alignment of action transcripts to video frames for weakly supervised learning, the training of the system, and a discussion of the convergence behavior and the respective stop criterion. An extensive experimental evaluation is provided in Section 6.4.

6.2 Task Description

We address the same action segmentation task that has been addressed by the previous fully supervised approach and that is outlined in Section 3.1.2. In contrast to fully supervised action detection or segmentation approaches, however, weakly supervised learning is based on an ordered list of the actions occurring in the video rather than on frame-level annotation. The difference between the previously addressed case of full, frame-level annotation and weak annotations by action transcripts as proposed in this chapter is shown in Figure 6.1. A video of the activity *making cereals* might consist of taking a bowl, pouring cereals in it, adding milk, and stirring cereals and milk. In a fully supervised task a temporal annotation of each action start and end time would be available for training, e.g. in form of

```
1 - 120:  take_bowl
```



(a) Fully Supervised

take_bowl	pour_cereals	pour_milk	stir_cereals
frame 1-120	frame 121-399	frame 400-446	frame 447-577

(b) Weakly Supervised

`take_bowl, pour_cereals, pour_milk, stir_cereals`

Figure 6.1: The amount of supervision in (a) the fully supervised case and (b) the weakly supervised case. While in the fully supervised case, frame-level annotation is provided, in the weakly supervised case only an action transcript is given and no explicit frame-level annotation is available.

```
121 - 399:  pour_cereals
400 - 446:  pour_milk
447 - 577:  stir_cereals.
```

In the weakly supervised setup, all videos are just labeled with their ordered action sequence given as

```
take_bowl, pour_cereals, pour_milk, stir_cereals.
```

Considering that the goal is to find a segmentation of the video that specifies the number of action instances N, their class labels $\mathbf{c}_1^N$, and the segment lengths $\mathbf{l}_1^N$, in the fully supervised case all information that is to be inferred later is given during training, i.e. the full segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$ is available for each training video. In the weakly supervised case, on the contrary, the length information is not available during training and the annotation for each training video only provides the number of segments and their labels $(N, \mathbf{c}_1^N)$.

As this information is available for each video and as long as all actions appear at least once in different contexts, it is possible to infer the related action boundaries without frame-based ground truth information, in our case by choosing the related action representations in a way that they maximize the probability that the sequences were generated by the respective models.

Note that this also formulates the necessary preconditions of the overall system, namely the fact that it needs the order of all actions as they appear in the sequences of the training set and that all actions need to appear at least with two different predecessors and successors. This constraint is necessary as we want to maximize the probability that the sequences of

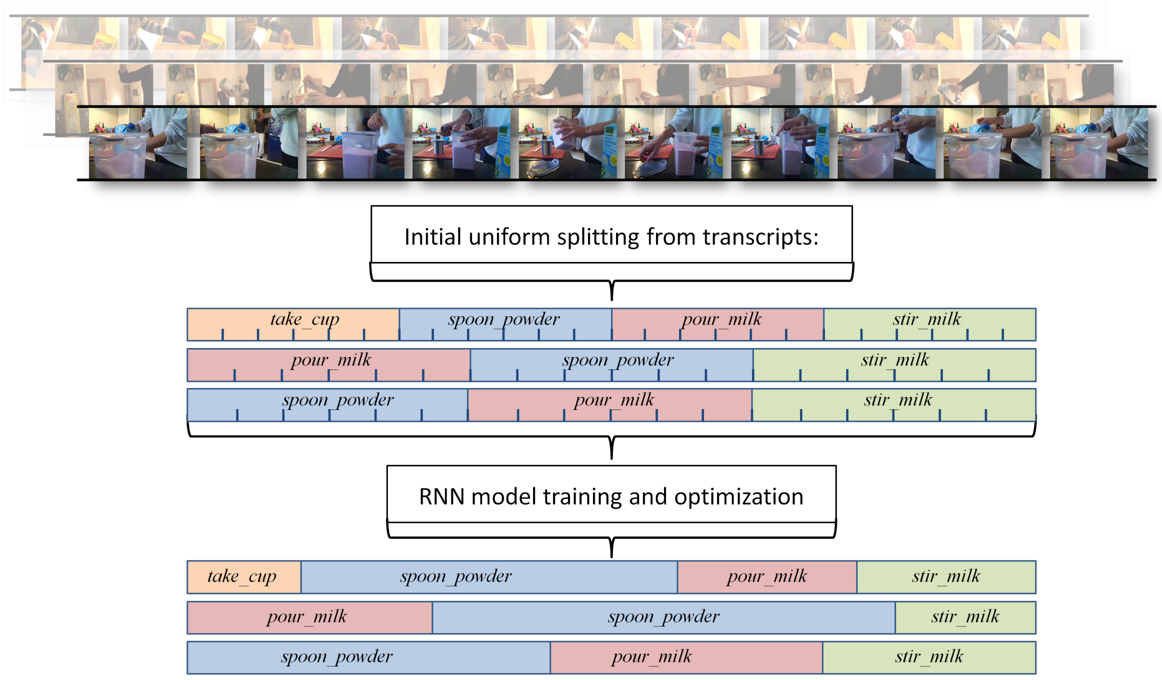


Figure 6.2: Overview of the proposed weak learning system. Given a list of ordered actions for each video, an initial segmentation is generated by uniform segmentation. Based on this input information we iteratively train an RNN-based fine-to-coarse system to align the frames to the respective action.

the training data are generated by models trained on a set of boundary assumptions. If e.g. two actions always occur together and in the same order, the position of the boundary in between the two actions can not be learned as there is no hint in the training data of how the end of the first action or the start of the second action looks like. Only if one action appears in a different context, i.e. with different predecessors and successors, we are able to learn a reasonable visual model with respect to different boundaries.

6.3 Technical Details

We again follow the general framework proposed in Chapter 4,

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \left\{ p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) \cdot p(\mathbf{l}_1^N | \mathbf{c}_1^N) \cdot p(\mathbf{c}_1^N) \right\}, \quad (6.1)$$

but omit the length model $p(\mathbf{l}_1^N|\mathbf{c}_1^N)$, so the approach used in this chapter only features a visual model and a context model. Instead, we propose a HMM-based visual model that implicitly models length *(a)* via the number of HMM states and *(b)* via a length regularization on the HMM states.

As sequential actions are naturally composed of hierarchical movements and actions at different levels of temporal granularity, we follow the idea of a hierarchical action model and adapt it for the case of weak learning of human actions in video data.

At top level, we model each temporal sequence as a combination of basic actions. This can be an activity, as e.g. *making tea* which would be made up of the actions *take cup*, *add teabag* and *pour water*. Each of those actions is represented by a respective probabilistic graph model, in this case an HMM, which models each action as a combination of subactions. Intuitively, the idea of subactions is based on the fact that, e.g. an action such as *take cup* consists of multiple movements like *move hand towards cup*, *grab cup*, *move cup towards body* and *release cup*.

The proposed model captures those implicitly available but not explicitly annotated subactions as latent variables by the states in the HMM. In order to build the state graph, it is not necessary to know the true number or label of the possible subactions. Instead, we set the number of subactions relative to the length of the corresponding action and update this factor as part of the training. Thus subactions at the beginning of an action capture motion patterns typical for that phase, as e.g. for *take cup* the first subactions comprise elements such as *move hand towards cup*. To ensure the sequential peculiarity of human actions within the HMM, we use a feed-forward topology, allowing only self-transition or transitions to the next state. We also show that this characteristic can be further supported by introducing a state specific length model to regularize the duration of each state during inference.

In the following, we describe the proposed framework in detail, starting with the formal definition of the hierarchical action model. After that, we discuss the different elements of our model, the fine-grained subaction classification and the length regularizer in detail. Next, we describe the inference and training procedure for the weakly supervised case and close with a discussion of the chosen stop criterion.

6.3.1 Visual Model

For the fully supervised approach, we used a segment-level visual model, i.e. a model that is already trained on trimmed action segments that can be obtained from the ground truth annotation. For weakly supervised learning, such supervision is not available anymore. We therefore introduce a visual model that operates on a per-frame basis and can be trained given the transcripts only.

Formally, we can assume the training data is a set of tuples $(\mathbf{x}_1^T, \mathbf{c}_1^N)$, where $\mathbf{x}_1^T = (x_1, \dots, x_T)$ are framewise features of a video with T frames and $\mathbf{c}_1^N$ is an ordered sequence $(c_1, \dots, c_N)$ of actions occurring in the video, as defined in the previous chapters. Note that the goal is still to infer a complete segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$ but as supervision, only $(N, \mathbf{c}_1^N)$ are given and the ground truth lengths of the segments are unknown during training. When $\mathbf{c}_1^N$ is known, a segmentation of the video is defined by the mapping

$$n(t) : \{1, \dots, T\} \mapsto \{1, \dots, N\} \tag{6.2}$$

that has been introduced in Equation (4.5) and assigns an action segment index to each frame. While in the fully supervised case, this function has been given during training implicitly by $\mathbf{l}_1^N$, in the current weakly supervised setting, it needs to be learned. Initially, $n(t)$ can be modeled by a linear segmentation of the provided actions, see Figure 6.10a. Assuming that

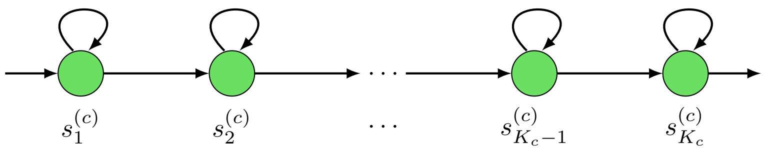


Figure 6.3: 0-1 HMM architecture for an action c. From a subaction state $s_i^{(c)}$, it is only possible to either stay in the state or proceed to the successor state $s_{i+1}^{(c)}$.

the frames are independent and only depend on their assigned action label $c_{n(t)}$ rather than on the complete transcript $\mathbf{c}_1^N$, the likelihood of a video sequence $\mathbf{x}_1^T$ given the action transcripts $\mathbf{c}_1^N$ is defined as

$$p(\mathbf{x}_1^T|\mathbf{c}_1^N) := \prod_{t=1}^{T} p\big(x_t|c_{n(t)}\big), \tag{6.3}$$

where $p(x_t|c_{n(t)})$ are probabilities of frame x_t being generated by the action $c_{n(t)}$.

The action classes usually describe longer, task-oriented procedures that naturally consist of more than one significant movement and we want to efficiently capture those characteristics. We model each action as a sequential combination of subactions. For each action class c, a set of subactions $s_1^{(c)}, \dots, s_{K_c}^{(c)}$ is defined. The number K_c is initially estimated by a heuristic and refined during the optimization process. Practically, this means that we subdivide the original long action classes into a set of smaller subactions. As subactions are obviously not defined by the given ordered action sequences, we treat them as latent variables that need to be learned by the model. In the following system description, we assume that the subaction frame boundaries are known, e.g. from previous iterations or from an initial uniform segmentation (see Figure 6.10b), and discuss the inference of concrete boundaries in Section 6.3.3.

In order to combine the fine grained subactions to action sequences, a hidden Markov model for each action c is defined. The HMM ensures that subactions only occur in the correct ordering, i.e. that $s_i^{(c)} \prec s_j^{(c)}$ for $i \leq j$. More precisely, let

$$\mathbf{s}_1^T = (s_1, \dots, s_T), \quad s_t \in \mathcal{S} = \{s_1^{(c_1)}, \dots, s_{K_{c_N}}^{(c_N)}\} \tag{6.4}$$

be an alignment from the set of possible subactions $\mathcal{S}$ to the time frames t. When going from one frame to the next, we only allow to assign either the same subaction or the next subaction, so if at frame t the assigned subaction is $s_t = s_i^{(c)}$, then at frame $t+1$ either $s_{t+1} = s_i^{(c)}$ or $s_{t+1} = s_{i+1}^{(c)}$, see Figure 6.3.

While the transition probabilities $p(s|\tilde{s})$ are learned during training for all $(s, \tilde{s})$ pairs that meet the 0-1-model condition, all other transition probabilities are set to zero. Exceptions are the probabilities to transition from an HMM end state $s_{K_{\tilde{c}}}^{(\tilde{c})}$ of some class $\tilde{c}$ to an HMM start state $s_1^{(c)}$ of another class c. Since we do not want the HMM to constrain transitions between action classes at all, we set such transition probabilities to one.

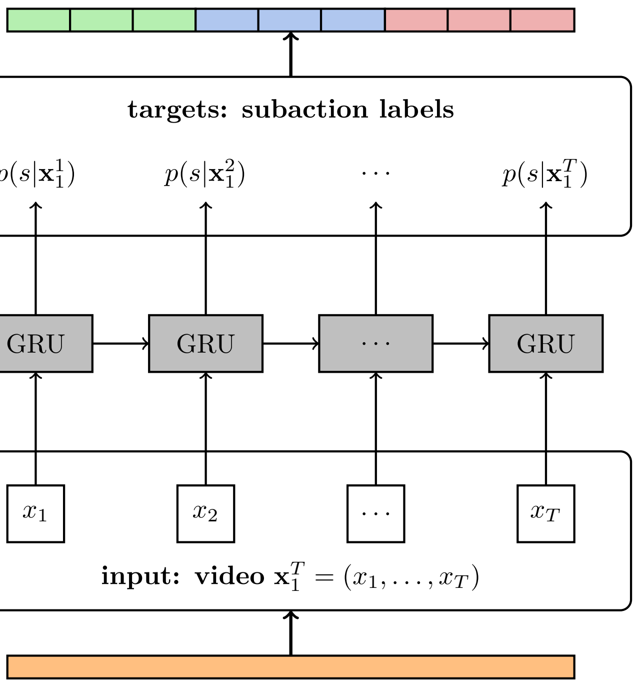


Figure 6.4: RNN using gated recurrent units with framewise video features as input. At each frame, the network outputs a probability for each possible subaction while considering the temporal context of the video by the preceding frames.

The likelihood of an action sequence $\mathbf{x}_1^T$ given the action transcripts $\mathbf{c}_1^N$ can then be expressed in terms of the HMM state alignment $\mathbf{s}_1^T$ as

$$p(\mathbf{x}_1^T|\mathbf{s}_1^T) := \prod_{t=1}^{T} p\big(x_t|s_t\big) \cdot p\big(s_t|s_{t-1}\big), \tag{6.5}$$

where $p(x_t|s)$ are probabilities computed by the fine-grained subaction model and $p(s_t|s_{t-1})$ is a HMM state transition probability. Equation (6.5) is the visual model in which the fine-grained subaction model still has to be defined.

Fine-grained Subaction Model

For the fine-grained subaction model $p(x_t|s_t)$ from Equation (6.5), we use an RNN with a single hidden layer of gated recurrent units which have been shown to work well in video classification (Ballas et al., 2016). The network is shown in Figure 6.4. For each frame, it predicts a probability distribution over all subactions, while the recurrent structure of the network allows to incorporate local temporal context. Since the RNN generates a posterior distribution $p(s|x_t)$ but our coarse model deals with subaction-conditional probabilities, we use Bayes' rule to transform the network output to

$$p(x_t|s) = \text{const} \cdot \frac{p(s|x_t)}{p(s)}, \tag{6.6}$$

and thus allow for a direct usage of the distributions generated by the recurrent network in Equation (6.5).

As recurrent neural networks are usually trained using back propagation through time (Werbos, 1990), which requires to process the whole sequence in a forward and backward pass and a video can be very long and may easily exceed $10,000$ frames, the computation time per minibatch can be extremely high. We therefore adapt the training procedure by using small chunks around each video frame. They can be efficiently processed with a reasonably large minibatch size in order to enable efficient RNN training on long videos. For each frame t, we create a chunk over $\mathbf{x}_{t-20}^{t}$ and forward it through the RNN. While this practically increases the amount of data that needs to be processed by a factor of 20, only short sequences need to be forwarded at once and we benefit from a high degree of parallelism and a comparably large minibatch size.

6.3.2 Context Model

While we used an m-gram in the previous chapter, we resort to a more restrictive context model here. Due to the dataset characteristics of Thumos, the de-facto benchmark for fully supervised action detection and segmentation, a limited context in form of an m-gram is sufficient. For many other datasets, however, context dependencies can be larger and span the complete action transcript. Moreover, as the visual model is much weaker for weakly supervised systems, a more restrictive context model can greatly help.

We present a context model here that is based on stochastic right-regular grammars. A stochastic right-regular grammar is a five-tuple

$$\Gamma = \big(H, \mathcal{C}, \mathcal{R}, h_{\$}, p(c|h)\big), \tag{6.7}$$

where H is the set of non-terminal symbols that in our case encode the possible action class histories and $h_{\$}$ is the start symbol. The set of terminal symbols is the set of action classes $\mathcal{C}$. The distribution $p(c|h)$ denotes the probability of producing a label $c \in \mathcal{C}$ given the context $h \in H$. A right-regular grammar further consists of a set of production rules $\mathcal{R}$ that are of the form

$$\tilde{h} \overset{p(c|\tilde{h})}{\longrightarrow} c\ h \qquad \text{or} \tag{6.8}$$

$$\tilde{h} \overset{p(c|\tilde{h})}{\longrightarrow} c, \tag{6.9}$$

where $\tilde{h}, h \in H$ are two non-terminal symbols, $c \in \mathcal{C}$ is an action class, and $p(c|\tilde{h})$ denotes the probability of a transition from the context $\tilde{h}$ to the context h when hypothesizing an action class c.

Note that every stochastic finite automaton can be represented as a right-regular grammar with the states being the non-terminal symbols H, the inputs being the action labels $\mathcal{C}$ and the start state being $h_{\$}$. The transition probabilities for transitioning from a state $\tilde{h}$ to a state h via class c correspond to the probabilities associated with the respective rule $\tilde{h} \overset{p(c|\tilde{h})}{\longrightarrow} c\ h$ and rules of the form $\tilde{h} \overset{p(c|\tilde{h})}{\longrightarrow} c$ are transitions into a unique end state. This in mind, it is worth mentioning that right-regular grammars can model arbitrary regular expressions and the previously used m-grams are an instance of such a right-regular grammar.

In this work, we use a simple kind of right-regular grammar that can produce exactly the action transcripts $\mathbf{c}_1^N$ that occur in the training set. We assign equal probability to all of these sequences. Figure 6.5 illustrates how the grammar is created from the training transcripts. Transcripts that share a common prefix can be represented by the same path in the resulting tree-like automaton and branch at the first point at which they differ. For each history of action labels, there is a non-terminal symbol. Assigning equal probability to each possible path is equivalent to setting all transition probabilities to 1.0. Note that a renormalization of the resulting distribution $p(c|h)$ is not necessary since the renormalization factor would not affect the maximizing arguments. This simple kind of grammar proves to be efficient for the datasets used in this work, however, the framework allows for more complex stochastic grammars in principle.

6.3.3 Inference

For inference, the goal is to find the best segmentation for a given video $\mathbf{x}_1^T$. To do so, we use a Viterbi based inference method based on the hidden Markov model formulation of our system. While previously, a Viterbi decoding over the action classes could be used, we now need to maximize over a sequence of subaction classes that implicitly defines the segmentation on action class level. The decoding is similar to the previously discussed simple Viterbi version from Chapter 4 and the modified version from Chapter 5 but requires some adjustments in order to work on subaction level.

Recall that we omit the length model in our model formulation here as the HMMs in the visual model implicitly model action lengths by the number of subactions or states per action class. We are therefore interested in finding the best segmentation as

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) \cdot p(\mathbf{c}_1^N) \Big\}. \tag{6.10}$$

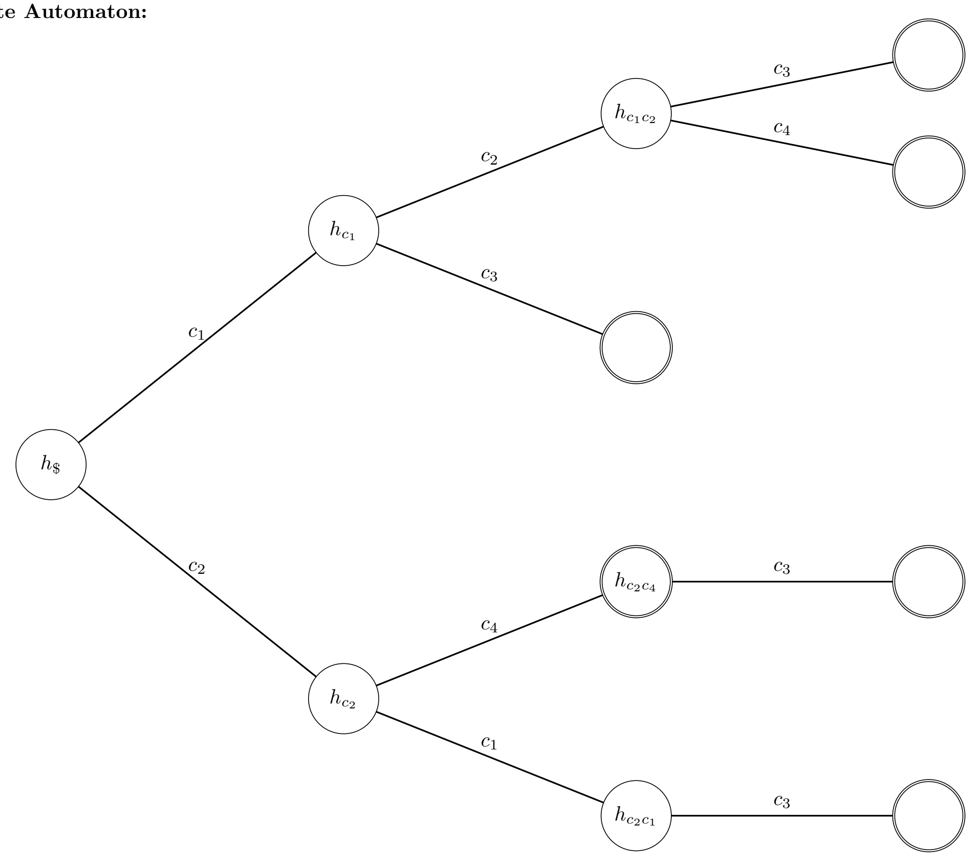
Before we can insert the actual models into the above equation, consider that the visual model $p(\mathbf{x}_1^T|\mathbf{s}_1^T)$ is conditioned on a state sequence $\mathbf{s}_1^T$ rather than on the class and length sequences $(\mathbf{c}_1^N, \mathbf{l}_1^N)$. Since each HMM state s_t in the state sequence $\mathbf{s}_1^T$ is associated with a concrete subaction $s_i^{(c)}$ for some action class c, the state sequence $\mathbf{s}_1^T$ defines a unique segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$. Let $\mathcal{E} : \mathbf{s}_1^T \mapsto (N, \mathbf{c}_1^N, \mathbf{l}_1^N)$ be an extractor function that maps from the state sequence to the underlying segmentation, see Figure 6.6 for an illustration.

The remaining obstacle is how to connect the context model $p(c|h)$ with the state sequence $\mathbf{s}_1^T$. Consider the segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$. This segmentation is only possible, if $\mathbf{c}_1^N$ can be generated by the grammar. Therefore, there must be a sequence $\mathbf{h}_1^N$ of non-terminal symbols such that for every $n \in \{1, \dots, N\}$, there is a production rule $h_{n-1} \rightarrow c_n\ h_n$ with probability greater than zero in the set of rules $\mathcal{R}$. According to the definition of our grammar, the last rule in the generation process is always of the form $h \rightarrow c$. For simplicity, we use an equivalent notation where the last applied production rule is of the form $h_{N-1} \rightarrow c_N\ h_N$ with a virtual end symbol $h_N = h_{\text{end}}$ that only occurs on right-hand-sides of production rules and

Transcripts:

(c_1, c_2, c_3)
(c_2, c_4, c_3)
(c_1, c_2, c_4)
(c_1, c_3)
(c_2, c_4)
(c_2, c_1, c_3)

Finite Automaton:



Resulting Rules:

$$h_\$ \xrightarrow{1.0} c_1\ h_{c_1}$$
$$h_\$ \xrightarrow{1.0} c_2\ h_{c_2}$$
$$h_{c_1} \xrightarrow{1.0} c_2\ h_{c_1c_2}$$
$$h_{c_1} \xrightarrow{1.0} c_3$$
$$h_{c_2} \xrightarrow{1.0} c_4$$
$$h_{c_2} \xrightarrow{1.0} c_4\ h_{c_2c_4}$$
$$h_{c_2} \xrightarrow{1.0} c_1\ h_{c_2c_1}$$
$$h_{c_1c_2} \xrightarrow{1.0} c_3$$
$$h_{c_1c_2} \xrightarrow{1.0} c_4$$
$$h_{c_2c_4} \xrightarrow{1.0} c_3$$
$$h_{c_2c_1} \xrightarrow{1.0} c_3$$

Figure 6.5: Generation of the right-regular grammar from the training transcripts. Left: transcripts given during training. Middle: (stochastic) finite automaton generated from the transcripts. Final states are indicated by a double line. Probabilities on the arcs have been omitted since we assign equal probability to each path in our grammar. Right: resulting rules of the right-regular grammar. Note that for our datasets, very simple grammars that generate exactly those transcripts that occur in the training data are sufficient. More complex models that also assign some probability mass to paths not seen in training can also be modeled with right-regular grammars; m-grams (cf. Chapter 5) are an example for such models.

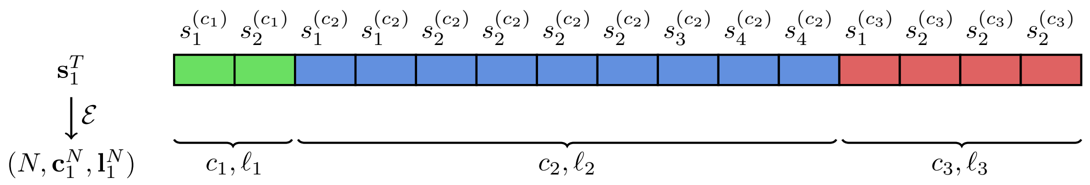

$s_1^{(c_1)}$ $s_2^{(c_1)}$ $s_1^{(c_2)}$ $s_1^{(c_2)}$ $s_2^{(c_2)}$ $s_2^{(c_2)}$ $s_2^{(c_2)}$ $s_2^{(c_2)}$ $s_3^{(c_2)}$ $s_4^{(c_2)}$ $s_4^{(c_2)}$ $s_1^{(c_3)}$ $s_2^{(c_3)}$ $s_2^{(c_3)}$ $s_2^{(c_3)}$
$\mathbf{s}_1^T$
$\mathcal{E}$
$(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$
c_1, ℓ_1
c_2, ℓ_2
c_3, ℓ_3

Figure 6.6: Example for the extractor function $\mathcal{E}$. For a given state sequence $\mathbf{s}_1^T$, $\mathcal{E}$ maps to the underlying segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$.

thus also does not allow for further production of new action classes. Inserting the visual model and the context model, Equation (6.10) can then be rewritten as finding the best state alignment that meets the above-mentioned requirements,

$$\hat{\mathbf{s}}_1^T = \underset{\substack{\mathbf{s}_1^T: \\ (N,\mathbf{c}_1^N,\mathbf{l}_1^N)=\mathcal{E}(\mathbf{s}_1^T), \\ \exists \mathbf{h}_1^N \forall n:\ h_{n-1} \rightarrow c_n h_n \in \mathcal{R}}}{\arg\max} \left\{ \prod_{t=1}^{T} p(x_t|s_t) \cdot p(s_t|s_{t-1}) \cdot \prod_{n=1}^{N} p(c_n|h_{n-1}) \right\} \tag{6.11}$$

and the optimal segmentation $(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}})$ is given by application of the extractor function $\mathcal{E}$ to the best HMM state sequence $\hat{\mathbf{s}}_1^T$.

In the following, we derive the recursive equations to efficiently find the optimal HMM state sequence $\hat{\mathbf{s}}_1^T$. As a first step, we remove the product over the N segments and rewrite Equation (6.11) as a product over the frames only,

$$\hat{\mathbf{s}}_1^T = \underset{\substack{\mathbf{s}_1^T: \\ (N,\mathbf{c}_1^N,\mathbf{l}_1^N)=\mathcal{E}(\mathbf{s}_1^T), \\ \exists \mathbf{h}_1^N \forall n:\ h_{n-1} \rightarrow c_n h_n \in \mathcal{R}}}{\arg\max} \left\{ \prod_{t=1}^{T} p(x_t|s_t) \cdot p(s_t|s_{t-1}) \cdot \left[p(c_{n(t)}|h_{n(t)-1})\right]^{[\![\exists c,\tilde{c}: s_t = s_1^{(c)} \wedge\ s_{t-1} = s_{K_{\tilde{c}}}^{(\tilde{c})}]\!]} \right\}, \tag{6.12}$$

where $[\![\cdot]\!]$ is again the index function and the term inside the index function is true if s_t is the first subaction $s_1^{(c)}$ for some action class c and at the previous frame, the state s_{t-1} was the last subaction state $s_{K_{\tilde{c}}}^{(\tilde{c})}$ of a preceding class $\tilde{c}$. In other words, the index function evaluates to true if t is the first time frame of a new action class, so the context model needs to be multiplied. Note that the function $n(t)$, which is used in the context model $p(c_{n(t)}|h_{n(t)})$, is available since it only depends on the lengths $\mathbf{l}_1^N$ which can be obtained by application of the extractor function to the state sequence $\mathbf{s}_1^T$.

Similarly to the derivation in the previous chapter, we focus on finding the probability of the best sequence, i.e. the max instead of the arg max, and introduce an auxiliary function $Q(\tau, s, h)$ that defines the best HMM state sequence up to time τ that ends in state s with context h,

$$Q(\tau, s, h) = \max_{\substack{\mathbf{s}_1^\tau: s_\tau = s, \\ (\eta, \mathbf{c}_1^\eta, \mathbf{l}_1^\eta) = \mathcal{E}(\mathbf{s}_1^\tau), \\ \exists \mathbf{h}_1^\eta \forall n:\ h_{n-1} \to c_n h_n \in \mathcal{R}, \\ h_\eta = h}} \Big\{ \prod_{t=1}^{\tau} p(x_t|s_t) \cdot p(s_t|s_{t-1}) \cdot \big[p(c_{n(t)}|h_{n(t)-1})\big]^{[\![\exists c, \tilde{c}: s_t = s_1^{(c)} \wedge\ s_{t-1} = s_{K_{\tilde{c}}}^{(\tilde{c})}]\!]} \Big\}. \tag{6.13}$$

Also similar to the derivation in the previous chapter, we isolate the last factor for time τ in the equation,

$$Q(\tau, s, h) = \max_{\substack{\mathbf{s}_1^\tau: s_\tau = s, s_{\tau-1} = \tilde{s} \\ (\eta, \mathbf{c}_1^\eta, \mathbf{l}_1^\eta) = \mathcal{E}(\mathbf{s}_1^\tau), \\ \exists \mathbf{h}_1^\eta \forall n:\ h_{n-1} \to c_n h_n \in \mathcal{R}, \\ h_\eta = h}} \Big\{ \Big[\prod_{t=1}^{\tau-1} p(x_t|s_t) \cdot p(s_t|s_{t-1}) \cdot \big[p(c_{n(t)}|h_{n(t)-1})\big]^{[\![\exists c, \tilde{c}: s_t = s_1^{(c)} \wedge\ s_{t-1} = s_{K_{\tilde{c}}}^{(\tilde{c})}]\!]} \Big] \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \cdot \big[p(c_{n(\tau)}|h_{n(\tau)-1})\big]^{[\![\exists c, \tilde{c}: s = s_1^{(c)} \wedge\ \tilde{s} = s_{K_{\tilde{c}}}^{(\tilde{c})}]\!]} \Big\}. \tag{6.14}$$

We now make a case distinction into the *within segment* case and the *between segment* case. In the within segment case, there is no new segment starting at time τ, so the index function in the last term of Equation (6.14) evaluates to zero and

$$\begin{aligned} Q(\tau, s, h) &= \max_{\tilde{s} \in \{s, \mathrm{pred}(s)\}} \max_{\substack{\mathbf{s}_1^{\tau-1}: s_{\tau-1} = \tilde{s} \\ (\eta, \mathbf{c}_1^\eta, \mathbf{l}_1^\eta) = \mathcal{E}(\mathbf{s}_1^{\tau-1}), \\ \exists \mathbf{h}_1^\eta \forall n:\ h_{n-1} \to c_n h_n \in \mathcal{R}, \\ h_\eta = h}} \Big\{ \Big[\prod_{t=1}^{\tau-1} p(x_t|s_t) \cdot p(s_t|s_{t-1}) \cdot \big[p(c_{n(t)}|h_{n(t)-1})\big]^{[\![\exists c, \tilde{c}: s_t = s_1^{(c)} \wedge\ s_{t-1} = s_{K_{\tilde{c}}}^{(\tilde{c})}]\!]} \Big] \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \Big\} \\ &= \max_{\tilde{s} \in \{s, \mathrm{pred}(s)\}} \big\{ Q(\tau - 1, \tilde{s}, h) \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \big\}, \end{aligned} \tag{6.15}$$

where we use that the inner maximization in the first line is $Q(\tau - 1, \tilde{s}, h)$ with the definition from Equation (6.13). The function $\mathrm{pred}(s)$ denotes the preceding subaction of s, i.e. if s is an instance of subaction $s_i^{(c)}$, $\mathrm{pred}(s) = s_{i-1}^c$. Recall that we use a 0-1 HMM, so within a segment, it is either possible to transition from the same subaction (loop) or from the preceding subaction, cf. Figure 6.3. In the within-segment case, s can not be an HMM start state, so $\mathrm{pred}(s)$ is always defined.

For the between segment case, the index function evaluates to one and the situation is a bit more involved. If a new segment starts at time τ, the extractor function returns different results for $\mathcal{E}(\mathbf{s}_1^\tau)$ and $\mathcal{E}(\mathbf{s}_1^{\tau-1})$. Therefore, it needs to be made sure that there is a rule from

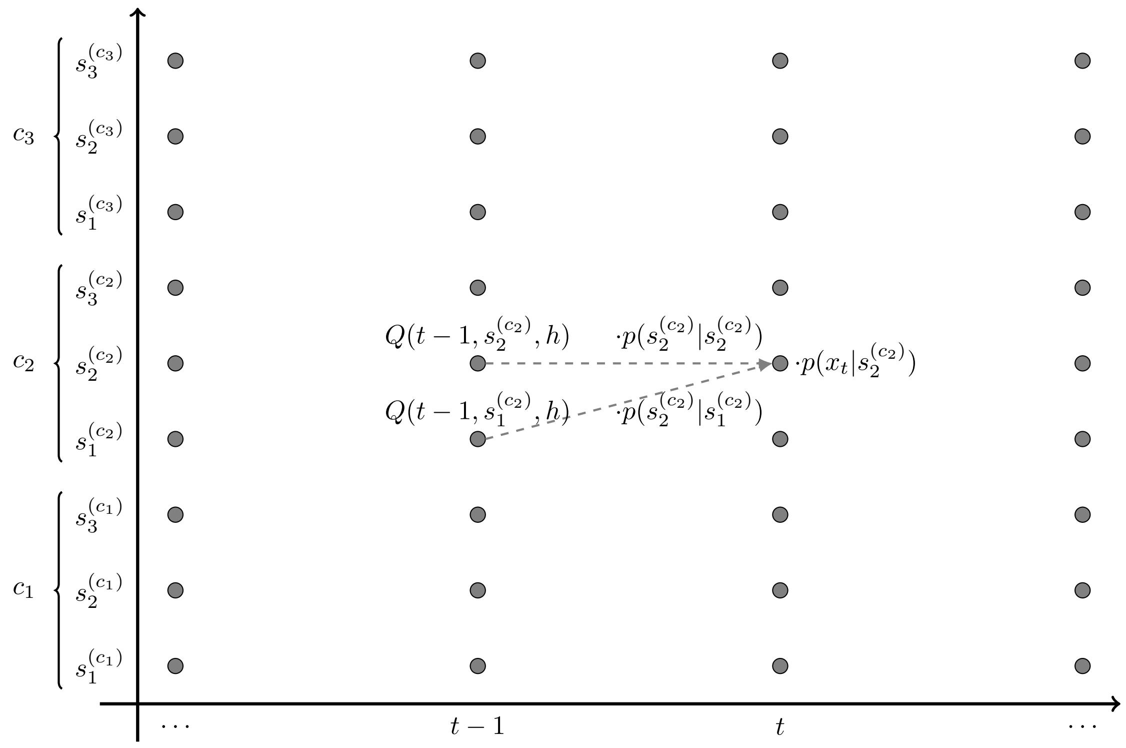
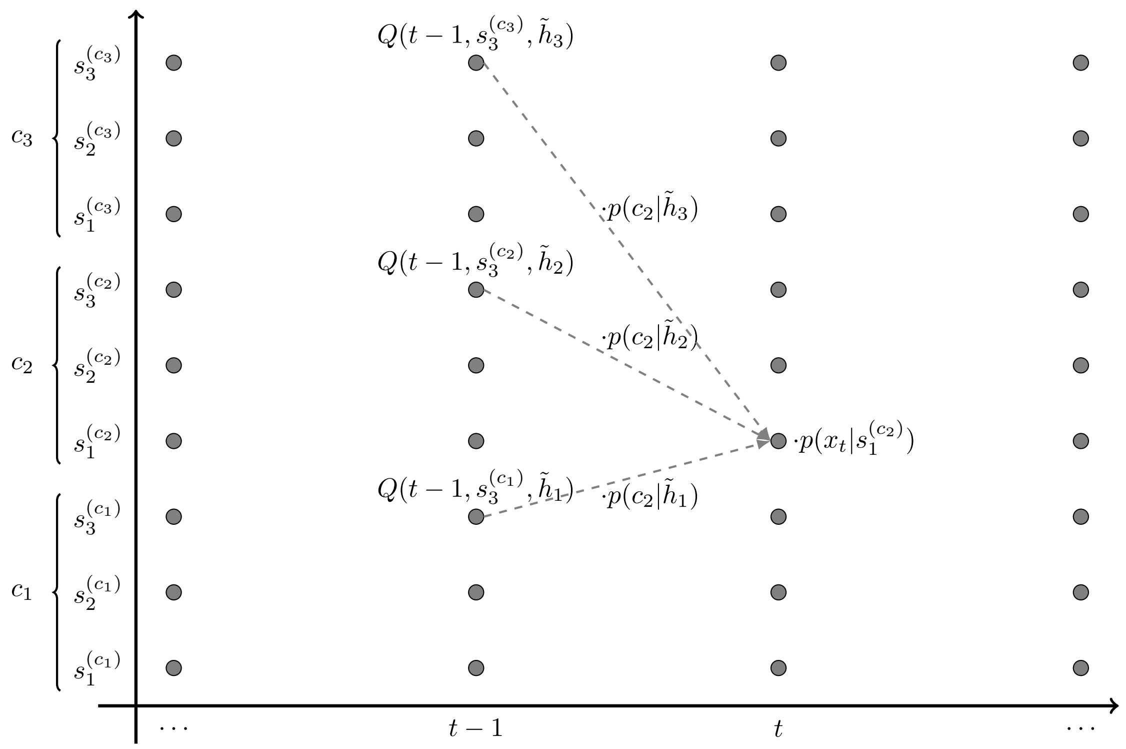


Figure 6.7: Viterbi decoding in the within segment case. For each state $s_i^{(c)}$ that is not an HMM start state, there are only two possible predecessor states, $s_i^{(c)}$ itself and $s_{i-1}^{(c)}$.

the last context $\tilde{h}$ available in the maximization over $\mathbf{s}_1^{\tau-1}$ that produces the class c and the new context h, i.e. there must be a rule $\tilde{h} \to c\ h$. Further, in the between segment case, s is always a start state of an subaction HMM, i.e. $s = s_1^{(c)}$ for some class c. Thus,

$$\begin{aligned} Q(\tau, s = s_1^{(c)}, h) = &\max_{\substack{\tilde{s},\tilde{h}: \\ \tilde{h}\to c\ h\in\mathcal{R}}} \max_{\substack{\mathbf{s}_1^{\tau-1}: s_{\tau-1}=\tilde{s} \\ (\eta,\mathbf{c}_1^\eta,\mathbf{l}_1^\eta)=\mathcal{E}(\mathbf{s}_1^{\tau-1}), \\ \exists \mathbf{h}_1^\eta \forall n:\ h_{n-1}\to c_n h_n \in\mathcal{R}, \\ h_\eta=\tilde{h}}} \Big\{ \Big[\prod_{t=1}^{\tau-1} p(x_t|s_t) \cdot p(s_t|s_{t-1}) \\ &\qquad \cdot \big[p(c_{n(t)}|h_{n(t)-1})\big]^{[\![\exists c,\tilde{c}: s_t=s_1^{(c)} \wedge\ s_{t-1}=s_{K_{\tilde{c}}}^{(\tilde{c})}]\!]} \Big] \\ &\qquad \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \cdot p(c|\tilde{h}) \Big\} \\ = &\max_{\substack{\tilde{s},\tilde{h}: \\ \tilde{h}\to c\ h\in\mathcal{R}}} \big\{ Q(\tau-1, \tilde{s}, \tilde{h}) \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \cdot p(c|\tilde{h}) \big\}. \end{aligned} \tag{6.16}$$

Equation (6.15) and (6.16) show that both, the within segment case and the between segment case at time τ are again recursively defined by the function Q at the preceding time $\tau - 1$. Both cases of the Viterbi decoding are illustrated in Figure 6.7 and Figure 6.8, respectively. Note that an HMM start state $s_1^{(c)}$ can be reached in two ways, either in the within segment



$Q(t-1, s_3^{(c_3)}, \tilde{h}_3)$
$\cdot p(c_2|\tilde{h}_3)$
$Q(t-1, s_3^{(c_2)}, \tilde{h}_2)$
$\cdot p(c_2|\tilde{h}_2)$
$\cdot p(x_t|s_1^{(c_2)})$
$Q(t-1, s_3^{(c_1)}, \tilde{h}_1)$
$\cdot p(c_2|\tilde{h}_1)$
c_3
c_2
c_1
$s_3^{(c_3)}$
$s_2^{(c_3)}$
$s_1^{(c_3)}$
$s_3^{(c_2)}$
$s_2^{(c_2)}$
$s_1^{(c_2)}$
$s_3^{(c_1)}$
$s_2^{(c_1)}$
$s_1^{(c_1)}$
$\cdots$
$t-1$
t
$\cdots$

Figure 6.8: Viterbi decoding in the between segment case. A transition into an HMM start state $s_1^{(c)}$ can be made from every HMM end state with context $\tilde{h}$ if there is a rule $\tilde{h} \rightarrow c\ h$ in the grammar. In case of a simple bigram, the contexts $\tilde{h}_1$, $\tilde{h}_2$, and $\tilde{h}_3$ are simply the classes c_1, c_2, and c_3 of the preceding segments. For grammars with a larger context, there may be multiple possible contexts for each HMM end state. In that case, the decoding would go over an additional third axis of nonterminal symbols which has been omitted in the plot.

case via a loop transition from itself, or in the between segment case via a transition from an HMM end state.

The probability of the best state alignment is given as

$$\max_{s \in \{s_{K_c}^{(c)} : c \in \mathcal{C}\}} Q(T, s, h_{\text{end}}), \tag{6.17}$$

where h_{end} is the above mentioned virtual end symbol of the grammar that never occurs on a left-hand-side of a rule. The maximization is carried out over all HMM states that are an end state for an action class c. In order to reconstruct the best state alignment, two traceback arrays A and B are used, indicating the best HMM state at the previous time and the best context at the previous time. For the within segment case, those best predecessors are

$$A(\tau, s, h) = \operatorname*{arg\,max}_{\tilde{s} \in \{s, \text{pred}(s)\}} \big\{ Q(\tau-1, \tilde{s}, h) \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \big\}, \tag{6.18}$$

$$B(\tau, s, h) = h. \tag{6.19}$$

Algorithm 6.1 Viterbi Decoding

```
1:  $Q(1:T,:,:) = 0$
2:  $Q(0,:,h_\$) = 1$
3:  for $t = 1,\dots,T$ do
4:    for $s \in \mathcal{S}$ do
5:      for $h : \exists c, h' :\ c = \text{class}(s) \wedge h' \to c\ h \in \mathcal{R}$ do
6:        if $\exists c :\ s = s_1^{(c)}$ then
7:          for $\tilde{h} : \tilde{h} \to c\ h \in \mathcal{R}$ do
8:            for $\tilde{s} : \exists \tilde{c} :\ \tilde{s} = s_{K_{\tilde{c}}}^{(\tilde{c})} \wedge \exists h' :\ h' \to \tilde{c}\ \tilde{h}$ do
9:              if $Q(t,s,h) < Q(t-1,\tilde{s},\tilde{h}) \cdot p(x_t|s) \cdot p(c|\tilde{h})$ then
10:               $Q(t,s,h) = Q(t-1,\tilde{s},\tilde{h}) \cdot p(x_t|s) \cdot p(c|\tilde{h})$
11:               $A(t,s,h) = \tilde{s}$
12:               $B(t,s,h) = \tilde{h}$
13:       else
14:         for $\tilde{s} \in \{s, \text{pred}(s)\}$ do
15:           if $Q(t,s,h) < Q(t-1,\tilde{s},h) \cdot p(x_t|s) \cdot p(s|\tilde{s})$ then
16:             $Q(t,s,h) = Q(t-1,\tilde{s},h) \cdot p(x_t|s) \cdot p(s|\tilde{s})$
17:             $A(t,s,h) = \tilde{s}$
18:             $B(t,s,h) = h$
19: return $Q, A, B$
```

For the between segment case, we have

$$A(\tau, s = s_1^{(c)}, h) = \arg\max_{\tilde{s}} \Big\{ \max_{\substack{\tilde{h}: \\ \tilde{h} \to c\ h \in \mathcal{R}}} \big\{ Q(\tau-1,\tilde{s},\tilde{h}) \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \cdot p(c|\tilde{h}) \big\} \Big\}, \tag{6.20}$$

$$B(\tau, s = s_1^{(c)}, h) = \arg\max_{\substack{\tilde{h}: \\ \tilde{h} \to c\ h \in \mathcal{R}}} \Big\{ \max_{\tilde{s}} \big\{ Q(\tau-1,\tilde{s},\tilde{h}) \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \cdot p(c|\tilde{h}) \big\} \Big\}. \tag{6.21}$$

Complexity

Algorithm 6.1 provides a pseudo code implementation of the Viterbi decoding. In order to compute the entries of Q, a nested loop over all time frames and all possible HMM states is necessary. For the third component, the history, only those non-terminals h need to be considered that can be reached by producing a class c the subaction s belongs to, i.e. only those h that occur together with c on the right-hand-side of some rule, see line five of the algorithm. For the between segment case (line eight to thirteen), a further optimization is not to maximize over all possible HMM states $\tilde{s} \in \mathcal{S}$ but only over those that can actually be a predecessor of s. In fact, if s is an HMM start state, $\tilde{s}$ can only be an end state of another action class, else the transition probability is zero. Note that the transition probability between all pairs of end and start states has been set to one, so the transition term $p(s|\tilde{s})$ can be omitted in the between segment case. The for loop in the within segment case also just requires to run over the two elements s and pred(s) due to the 0-1 HMM architecture.

Algorithm 6.2 Reconstruction of the best segmentation

```
1: states = [ ]
2: $s = \underset{\tilde{s} \in \{s^{(c)}_{K_c} : c \in \mathcal{C}\}}{\arg\max} \; Q(T, \tilde{s}, h_{\text{end}})$
3: $h = h_{\text{end}}$
4: for $t = T, \dots, 1$ do
5:     states.append($s$)
6:     $s = A(t, s, h)$
7:     $h = B(t, s, h)$
8: states = states.revert()
9: $(N, \mathbf{c}_1^N, \mathbf{l}_1^N) = \mathcal{E}$(states)
10: return $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$
```

Overall, the Viterbi decoding has an asymptotic runtime of $\mathcal{O}(T \cdot |\mathcal{S}| \cdot |\mathcal{R}|)$, i.e. it is linear in the size of the number of rules in the grammar, the number of HMM states, and the length of the video. The traceback of the best state alignment is straightforward and linear in the video length T, see Algorithm 6.2. Note that also the extractor function $\mathcal{E}$ can be computed in one pass over the state sequence and thus, can also be computed in linear time.

Inference for Action Alignment. We also apply our model to temporal action alignment, cf. Section 3.1.2. For this task, the sequence of ordered actions is not only given during training but also for inference. Thus, only the best alignment between video frames and action transcript needs to be inferred. This task can be easily addressed using the proposed inference method when the context model at inference time is a grammar that generates only one sequence, i.e. the given action transcript for the video.

6.3.4 Length Regularization

In order to avoid degenerate solutions, we further add a state specific length regularizer as an additional factor to our state model. The length regularizer serves in this case as a temporal decay model that rewards the model at the beginning of a new state to stay in this state for a certain amount of frames and punishes the model if it stays too long in the same state. The idea is motivated by the observation that, without length regularizer, the system tends to skip many states, only remaining in those states for one or two frames. An example of this behavior is shown in Figure 6.9. It is clear to see that the HMMs in this case do not model any temporal progression. To evaluate this behavior further, we counted the so called skip states, i.e. states which are only assigned to one frame, for the case without and with length regularizer. We observe that without length regularization, 74.6% of all processed states can be counted as skip states whereas the introduction of the length regularizer, e.g. for the best performing configuration reported in Section 6.4.4, reduces the amount of skip states to 61.9%.

Formally, the length regularizer is a function of the duration of a state s at position t in the overall state-to-frame alignment, i.e. a function of the length $\ell(s_t)$ that captures how long

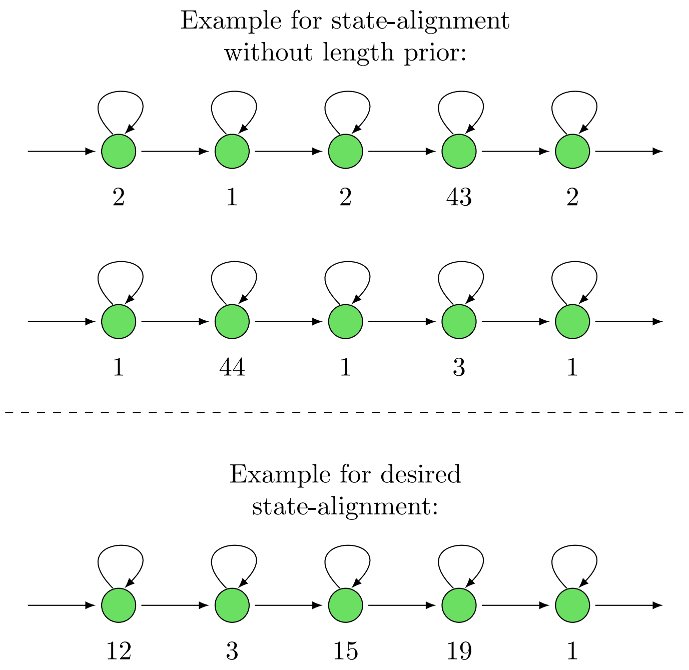

Example for state-alignment without length prior:
2
1
2
43
2
1
44
1
3
1
Example for desired state-alignment:
12
3
15
19
1

Figure 6.9: Example of state alignment for two instances of the same action as they are usually produced by the system without length regularization and of an instance showing the intended state alignment. In the first two cases, the HMM does not model the temporal progression but rather uses the subaction states to distinguish between different action appearances.

the model already remained in one state at time t. The length function is defined recursively as

$$\ell(s_t) = \begin{cases} \ell(s_{t-1}) + 1, & \text{if } s_t = s_{t-1}, \\ 1, & \text{otherwise.} \end{cases} \tag{6.22}$$

The regularizer $r(\ell(s_t)|s_t)$ models a decay factor based on the mean length $\text{len}(s_t)$ of the respective state. The mean length is given by the average length of each state computed as

$$\text{len}(s) = \frac{\text{number of frames aligned to } s}{\text{number of } s\text{-instances}}. \tag{6.23}$$

In Section 6.4.4, we analyze four different kinds of length regularizers, one with a strict upper bound on the length, a linearly decaying model, and one-sided Poisson and Gaussian distributions.

Decoding with Length Regularization

We incorporate the length regularizer $r(\ell(s_t)|s_t)$ into the visual model, so that the length-regularized visual model is

$$p(\mathbf{x}_1^T|\mathbf{s}_1^T) = \prod_{t=1}^{T} p(x_t|s_t) \cdot p(s_t|s_{t-1}) \cdot r(\ell(s_t)|s_t). \tag{6.24}$$

The updated recursive equations for the Viterbi decoding are then

$$Q(t,s,h) = \max_{\tilde{s}\in\{s,\mathrm{pred}(s)\}} \big\{Q(\tau-1,\tilde{s},h) \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \cdot r(\ell(s_t)|s_t)\big\} \tag{6.25}$$

for the within segment case and

$$Q(t,s=s_1^{(c)},h) = \max_{\substack{\tilde{s},\tilde{h}:\\ \tilde{h}\to c\ h\in\mathcal{R}}} \big\{Q(\tau-1,\tilde{s},\tilde{h}) \cdot p(x_\tau|s) \cdot p(s|\tilde{s}) \cdot r(\ell(s_t)|s_t) \cdot p(c|\tilde{h})\big\} \tag{6.26}$$

for the between segment case.

Due to the per-frame application of the length regularization, it needs to be a modification of a typical length distribution $p(\ell|s_t)$. Specifically, if $p(\ell|s_t)$ models the probability of s_t being ℓ frames long, the regularizer $r(\ell(s_t)|s_t)$ needs to fulfill

$$\prod_{i=1}^{\ell} r(\ell(s_{t-i+1})|s_{t-i+1}) = p(\ell|s_t). \tag{6.27}$$

In order to achieve this, the framewise length regularizer is defined as

$$r(\ell(s_t)|s_t) = \begin{cases} \frac{p(\ell(s_t)|s_t)}{p(\ell(s_{t-1})|s_{t-1})}, & \text{if } s_t = s_{t-1}, \\ p(\ell=1|s_t), & \text{otherwise.} \end{cases} \tag{6.28}$$

Particularly that means that the length model $p(\ell(s_{t-1})|s_{t-1})$ used at time $t-1$ is removed and replaced with the length model $p(\ell(s_t)|s_t)$ at time t if the HMM state did not change.

While the time complexity of the Viterbi decoding is maintained by the proposed length regularization, the resulting Viterbi path does not necessarily yield the globally best segmentation but only an approximation: while the regularizer is multiplied on a per-frame basis, it still depends on the duration the HMM already stayed in a specific state. The Viterbi decoding, however, only optimizes based on the results from the previous frame and does not maximize over the duration of a state.

Moreover, the underlying length distribution $p(\ell|s_t)$ must be monotonously deceasing. To illustrate why, consider a path that, at time t, has just changed to state s. A non-monotonous function such as a classical Poisson distribution would penalize such a path strongly, although the path may turn out very good if it stays in state s in the future. Using monotonous length models avoids this problem and only paths that are in a certain state s for too long are penalized. An evaluation of possible monotonous length models can be found in Section 6.4.4, where we analyze four different models, namely a box function, a linear decreasing function, a half Poisson, and a half Gaussian distribution.

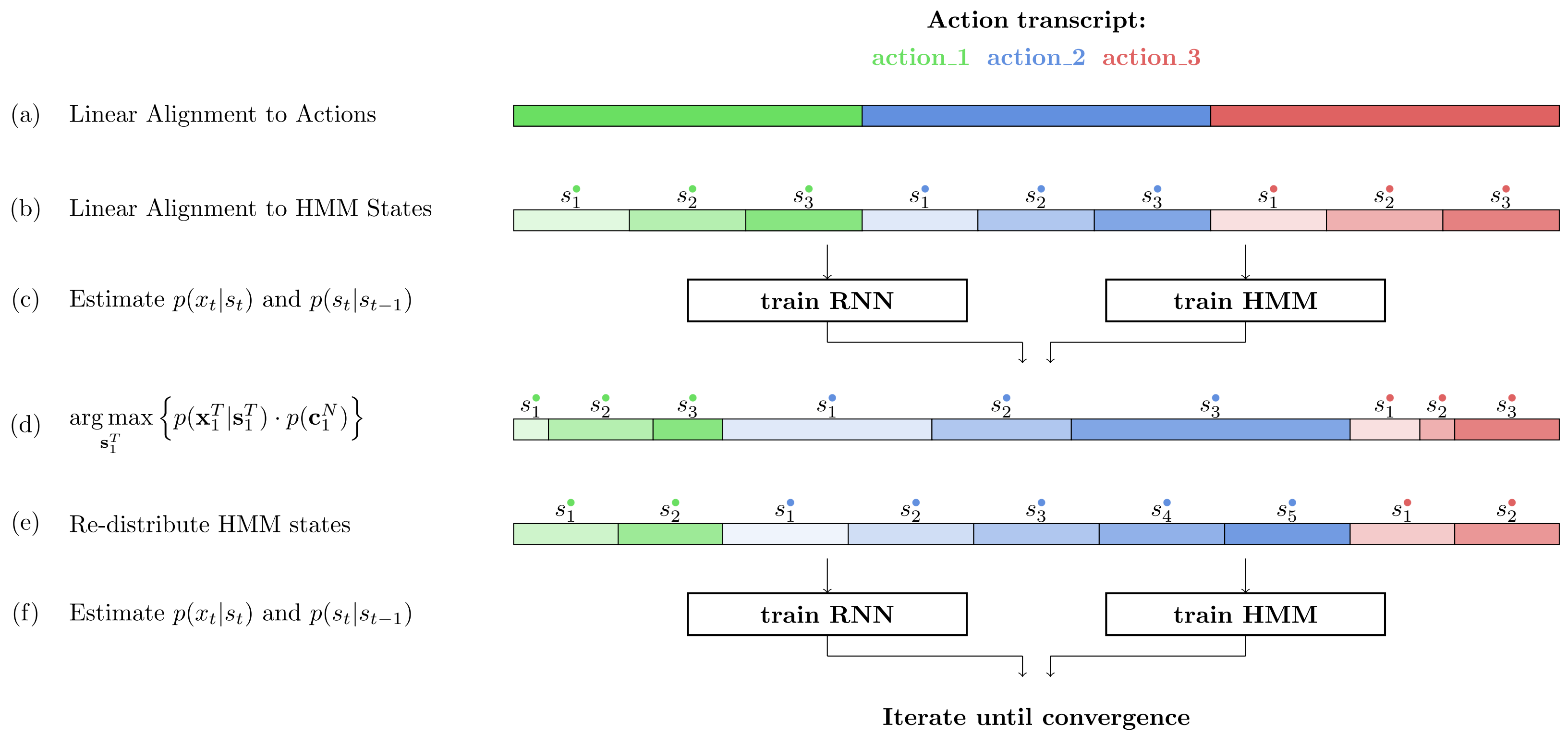

Action transcript:
action_1 action_2 action_3
(a) Linear Alignment to Actions
(b) Linear Alignment to HMM States
s_1 s_2 s_3 s_1 s_2 s_3 s_1 s_2 s_3
(c) Estimate $p(x_t|s_t)$ and $p(s_t|s_{t-1})$
train RNN
train HMM
(d) $\underset{\mathbf{s}_1^T}{\arg\max} \left\{ p(\mathbf{x}_1^T|\mathbf{s}_1^T) \cdot p(\mathbf{c}_1^N) \right\}$
s_1 s_2 s_3 s_1 s_2 s_3 s_1 s_2 s_3
(e) Re-distribute HMM states
s_1 s_2 s_1 s_2 s_3 s_4 s_5 s_1 s_2
(f) Estimate $p(x_t|s_t)$ and $p(s_t|s_{t-1})$
train RNN
train HMM
Iterate until convergence

Figure 6.10: Training process of our model. Initially, each action is modeled with the same number of subactions and the video is linearly aligned to these subactions. Based on this alignment, the RNN is trained and used in combination with the HMMs to realign the video frames to the subactions. Eventually, the number of subactions per action is reestimated and the process is iterated until convergence.

6.3.5 Training

The training of the model is done iteratively, altering between the recurrent neural network and the hidden Markov model training, and the alignment of frames to subactions via the hidden Markov model. The whole process is illustrated in Figure 6.10. We start with a linear segmentation and alignment of all training videos, train the respective RNN and HMM models and run an inference with the trained models which results in new frame boundaries for each action. We then redistribute the HMM states according to the new segmentation and repeat the training procedure several times until convergence is reached. In the following, we describe the two relevant steps, the initialization as well as the iterative training procedure in detail.

Initialization. Each video is divided into N segments of equal size, where N is the number of action instances in the transcript (Figure 6.10a). Each action segment is further subdivided equally across the subactions (Figure 6.10b). Note that this defines the alignment $\mathbf{s}_1^T$ between frames and subactions. Additionally, each subaction should cover m frames of an action on average. We fix m to 10 frames per subaction. Thus, the initial number of subactions for each action is

$$K_c = \frac{\text{total number of frames}}{\text{total number of action instances} \cdot m}. \tag{6.29}$$

Hence, initially each action is modeled with the same number of subactions. This can change during the following iterative optimization.

Training. The fine-grained RNN and the HMM are trained with the current alignment $\mathbf{s}_1^T$ as ground truth (Figure 6.10c). The RNN training follows the procedure outlined in Section 3.5. For the HMM, the transition probabilities $p(s|\tilde{s})$ are estimated as relative frequencies of the alignments $\mathbf{s}_1^T$ of each training video. If a length regularizer is used, the mean lengths for a state s are also estimated based on the alignments $\mathbf{s}_1^T$.

Using the updated RNN and HMM probabilities, the model from Equation (6.11) is applied to the training videos and a new alignment of frames to subactions (Figure 6.10d) is inferred using the inference algorithm described in Section 6.3.3. Note that during training, a context model that generates the provided action transcript $\mathbf{c}_1^N$ as the only possible path is used.

Reestimation. Once the realignment is computed for all training videos, the new average length of each action is computed and the number of subactions is re-estimated based on the updated average action lengths, so that there are $K_c = \text{len}(c)/m$ subactions for action c. The new subactions are again uniformly distributed among the frames assigned to the action (Figure 6.10e). This results in new state alignments $\mathbf{s}_1^T$ for each training video and the training process starts over from this point on. As the mapping for each iteration requires a different number of outputs for the RNN, we train a new RNN model for each iteration from scratch, initialized with random weights.

Context Model. During training, a grammar that only generates the action transcript of the video to be aligned is used. For inference, however, a general grammar that can produce various action transcripts is required. We learn a grammar that can produce exactly the sequences that occur during training. The probability for each rule in the grammar is set to one, making each action transcript equally likely.

Stop Criterion. As the system iteratively approximates the optimal action segmentation on the training data, we define a stop criterion based on the overall amount of frame labels changed from one iteration to the succeeding one. Overall, training is stopped if less than 5% of the frames are assigned a new label or a maximum of 15 iterations is reached.

6.4 Experiments

We first evaluate the performance of the different components of our system, namely the GRU based classification, the subaction modeling, and the length regularizer. We evaluate all tasks on the test set of the Breakfast dataset and report results as frame accuracy. The system is iterated until the stop criterion as described in Section 6.3.5 is reached.

6.4.1 Setup

We evaluate the proposed approach on two different datasets, Breakfast and Hollywood Extended, see Chapter 2 for a detailed description. Both datasets are standard benchmarks for weakly supervised action segmentation and are used in various other works on the topic (Kuehne et al., 2017; Huang et al., 2016). Note that in contrast to Thumos, where each video mainly consists of alternations between background and action instances of a single class, on Breakfast and Hollywood Extended, more complex action transcripts occur, requiring a grammar based context model rather than an m-gram that captures only short range action context. For both datasets we use framewise Fisher vectors of improved dense trajectories as described in Section 3.4.4. For the action model, we fix the number of hidden units in the RNN to 64. During training, no frame-level ground truth is used at all. The only annotation are the action transcripts $\mathbf{c}_1^N$ for each video.

We address the tasks of temporal action segmentation and temporal action alignment. For the temporal segmentation task, we report frame accuracy for Breakfast and the Jaccard index as intersection over union (IoU), see Section 3.2 for details. For action alignment, we follow the protocol of Bojanowski et al. (2014) and report intersection over detection (IoD).

6.4.2 Evaluation of the GRU-based Model

First, we evaluate the influence of the proposed fine grained RNN modeling. In order to analyze the capability of capturing temporal context with the recurrent network, we compare it to a system where a multilayer perceptron (MLP) is used instead. The MLP only operates on frame level and does not capture temporal context as there is no recurrent connection involved. In order to provide a fair comparison to the recurrent model, we setup the MLP with a single hidden layer of rectified units such that it has the same number of parameters as the recurrent network. We also look at the performance of standard GMM models, as they would be usually used in the context of HMMs. In this case, we follow the setup as described by Kuehne et al. (2016), using a single Gaussian distribution for each state of the model.

Breakfast	Iter 1	Iter 2	Iter 3	Iter 4	Iter 5
GMM w/o reest.	15.3	23.3	26.3	27.0	26.5
MLP w/o reest.	22.4	24.0	23.7	23.1	20.3
GRU w/o reest.	25.5	29.1	28.6	29.3	28.8

Table 6.1: Results for temporal action segmentation with the GRU-based model compared to an MLP-based model and a GMM over five iterations. It shows that the MLP and GMM are outperformed by the GRU-based model. Additionally the MLP-based model quickly starts to overfit whereas the GRU oscillates at a constantly higher level.

Breakfast	Accuracy (Mof)
GRU no subactions	22.4
GRU w/o reestimation	28.8
GRU + reestimation	33.3
GRU + GT length	51.3

Table 6.2: Results for temporal action segmentation on the Breakfast dataset comparing accuracy of the proposed system (GRU + reestimation) to the accuracy of the same architecture without subactions (GRU no subactions) and to the architecture with subclasses but without reestimation.

For this evaluation, we use a simplified version of the system without subaction reestimation or length regularization to achieve comparable results after each iteration. We show results for the first five iterations in Table 6.1. It becomes clear that GRUs outperform MLPs and GMMs, starting with 25.5% for the initial recognition, and reaching up to 29.3% after the fourth iteration. The MLP baseline stays continuously below this performance. Thus, it can be assumed that the additional information gained by recurrent connections in this context supports classification. One can further see that the MLP reaches its best performance after the second iteration and then continuously decreases, whereas the GRU begins to oscillate around 29%, hinting that the MLP also starts to overfit at an earlier stage compared to the GRU. The GMMs are also performing better than the MLP but, with a maximum of 27.0%, do not reach the performance of the GRU.

6.4.3 Analysis of the Subaction Modeling

Second, we investigate the properties of the proposed subaction modeling. We therefore compare the proposed system with the results of the same setting without further subdividing actions into subactions (GRU no subactions, Table 6.2). Additionally, we regard results of the system without reestimation of subactions during optimization (GRU w/o reestimation, Table 6.2). For the system without reestimation, we follow the initial steps as shown in Figure 6.10, thus, we linearly segment the videos according to the number of actions, generate an

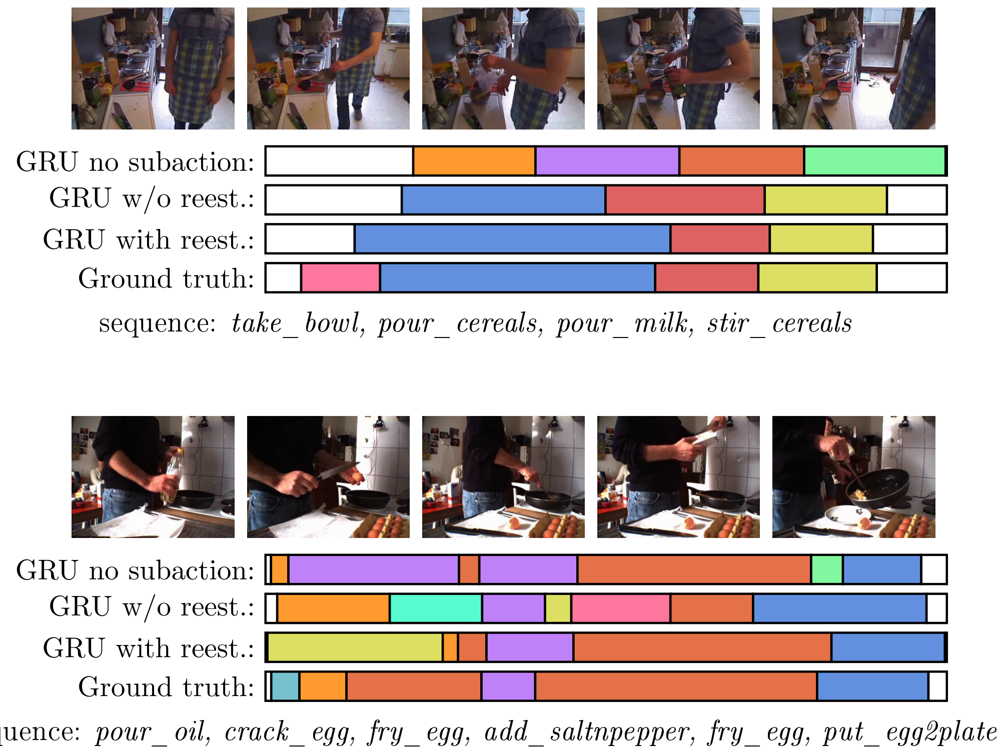


Figure 6.11: Example of temporal action segmentation for two samples from the Breakfast dataset showing the segmentation result for *preparing cereals* (top) and *preparing friedegg* (bottom). Although the actions are not always correctly detected, there is still a reasonable alignment of detected actions and ground truth boundaries.

initial subaction alignment, train the respective subaction classes, and realign the sequence based on the RNN output. But, opposed to the setup with reestimation, we omit the step of reestimating the number of subclasses and the following alignment. Instead, we just use the output of the realignment (see Figure 6.10d) to retrain the classifier and iterate the process of training, alignment, and re-training. Thus, the number of subclasses is constant and the coarse model is not adapted to the overall estimated length of the action class.

Finally, we compare to an approach in which we use the ground truth boundaries to compute the mean length of an action class and set the number of subactions based on the mean ground truth length (GRU + GT length, Table 6.2). Here, all action classes are still uniformly initialized, but longer action classes are divided into more subactions than shorter ones. We include this scenario as it models the performance in case that the optimal number of subaction classes would be found. We again use a simplified version of the system without length regularization to achieve comparable results.

Table 6.2 shows that the performance without subactions is significantly below all other configurations, supporting the idea that subaction modeling in general helps recognition in this scenario. The model with subactions, but without reestimation, improves over the single class model, but is still below the system with subaction reestimation. Compared to that, the

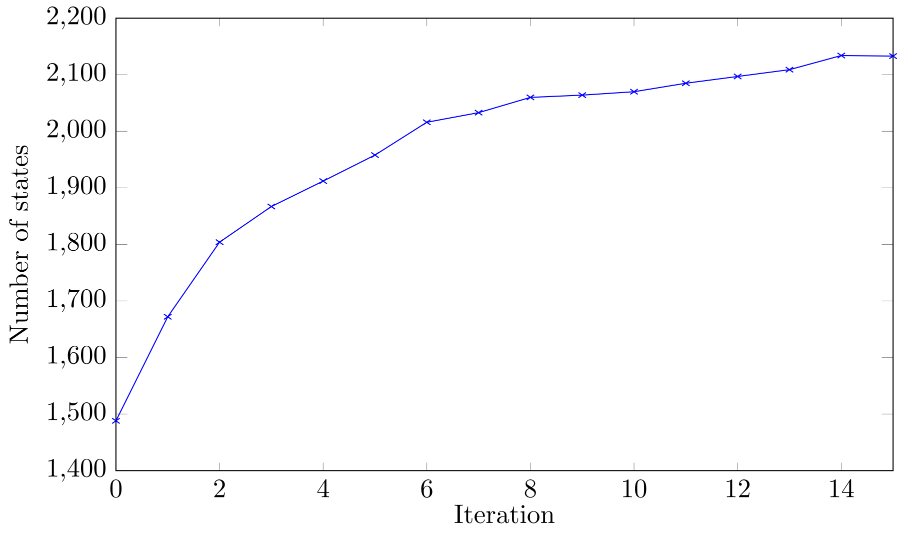


Figure 6.12: Evolution of the number of states for the model with state reestimation. The number of states increases in the first five iterations and converges after ten iterations.

model with subaction reestimation performs 5% better. We ascribe the performance increase of the reestimated model to the fact that a good performance is highly related to the correct number of subactions, thus to a good length representation of the single actions. The impact of the number of subactions becomes clear, when considering the results when the ground truth action lengths are used. The performance of the same system, just with different numbers of subactions, increases by almost 20%. This effect becomes also visible in the qualitative results as shown in Figure 6.11. Comparing the results of the three configurations – without subactions, without reestimation, and with reestimation – to the ground truth annotations, it shows that, although the overall sequence is not always correctly inferred by the models, the system with reestimation finds a good alignment compared to the ground truth frame boundaries. We also investigate how the overall number of subactions changes by reestimating after each iteration as shown in Figure 6.12. It becomes visible that the number of subactions mainly increases in the beginning and starts to converge after five to ten iterations.

6.4.4 Analysis of Length Regularization

In a next step, we analyze the impact of the length regularization on the overall system. Recall the definition of the regularizer r from Equation (6.28),

$$r(\ell(s_t)|s_t) = \begin{cases} \frac{p(\ell(s_t)|s_t)}{p(\ell(s_{t-1})|s_{t-1})}, & \text{if } s_t = s_{t-1}, \\ p(\ell = 1|s_t), & \text{otherwise.} \end{cases} \tag{6.30}$$

We examine four kinds of length distributions for $p(\ell|s_t)$ here. Note that we abbreviate $\ell(s_t)$ by ℓ_t for better readability. Also note that each distribution is normalized in a way that

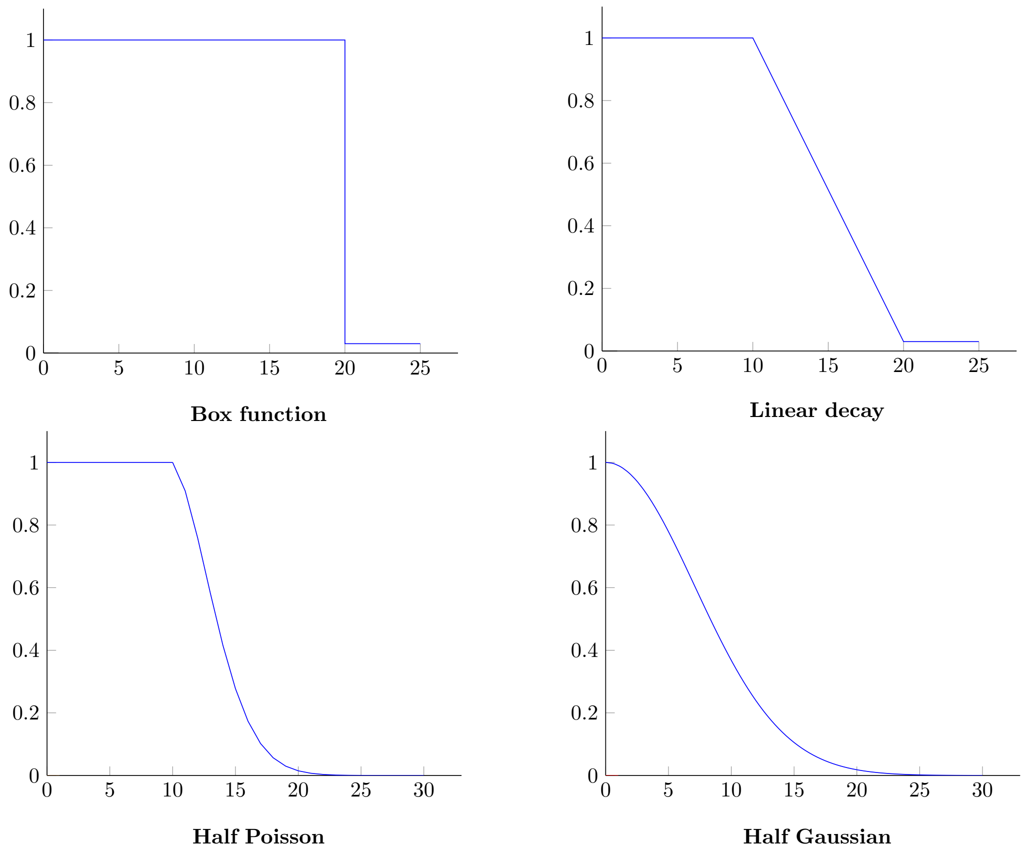


Figure 6.13: Overview of evaluated length models showing a simple box function, a linear decay function, a half Poisson decay and a half Gaussian function for a subaction with a mean length of 10 frames.

$\max\{p(\ell|s)\} = 1$, i.e. for each of the monotonically decreasing functions, the highest value is one. Although the models are not a strict probability distribution anymore, the normalization simplifies the formulas and sums up to a constant in the Viterbi decoding, not affecting the overall outcome.

The **box model** is defined as

$$p(\ell_t|s_t) = \begin{cases} 1, & \ell_t \leq 2 \cdot \text{len}(s_t), \\ \varepsilon, & \ell_t > 2 \cdot \text{len}(s_t), \end{cases} \tag{6.31}$$

and is considered as the basic representation of a length model. In this case the length regularizer does not influence the inference up to the point that twice the mean length of the respective state is reached. After that, the overall probability is multiplied with a given ε, so that the respective state is unlikely to be used any further.

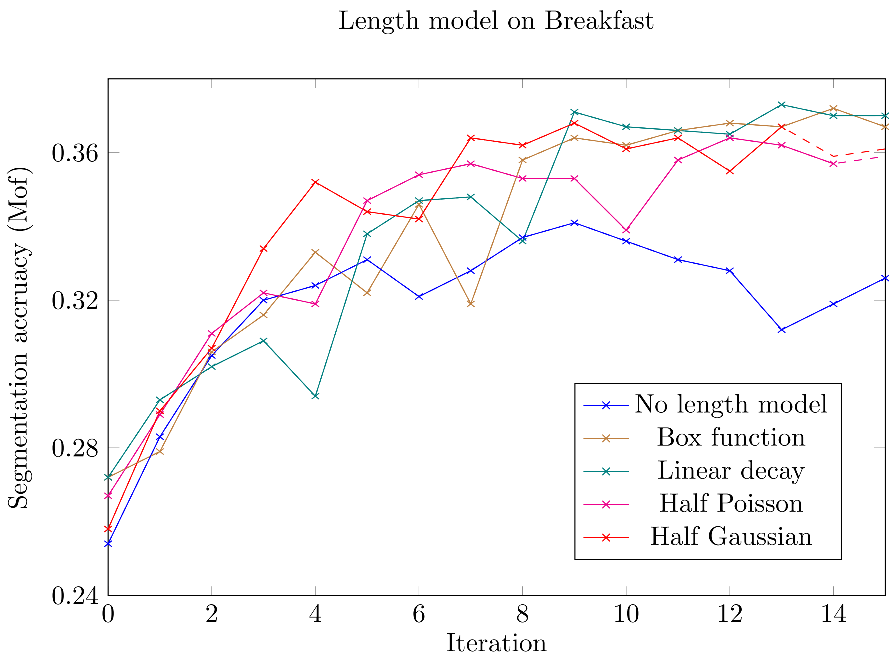


Figure 6.14: Results for temporal segmentation with different length models on the Breakfast dataset over 15 iterations. Solid lines show the results until the proposed stop criterion is reached. Dashed lines show the results after the stop criterion.

The **linear decay model**,

$$p(\ell_t|s_t) = \begin{cases} 1, & \ell_t \leq \text{len}(s_t), \\ 1 - \frac{\ell_t - \text{len}(s_t)}{\text{len}(s_t)}, & \ell_t > \text{len}(s_t) \wedge \ell_t < 2 \cdot \text{len}(s_t), \\ \varepsilon, & \ell_t \geq 2 \cdot \text{len}(s_t), \end{cases} \tag{6.32}$$

can be seen as an extension of the box function. Here, the length model is fix up to the point that the mean length of the respective state is reached. Then, the length model linearly decreases, punishing longer states more than shorter ones. After twice the mean length is reached, the probability is set to ε, so that the respective state is not used anymore.

The **half Poisson model** is constant on the left of the mode and a Poisson distribution on the right of the mode,

$$p(\ell_t|s_t) = \begin{cases} 1, & \ell_t \leq \text{len}(s_t), \\ \text{const} \cdot \frac{\text{len}(s_t)^{\ell_t}}{\ell_t!} e^{-\text{len}(s_t)}, & \ell_t > \text{len}(s_t). \end{cases} \tag{6.33}$$

The half Poisson model in the here proposed case also starts with a plateau and is fix up to the point that the mean length of the respective state is reached. After that we consider the right-half of the Poisson distribution of the respective state. We choose this combination as we want to model the discrete distribution of state lengths, and at the same time, ensure a monotonically decreasing function. For the half Poisson, const is a normalization factor such

Length model	Breakfast *frame acc.*	Hollywood Extended *Jacc. Idx*
No length model	32.6	11.5
Box function	36.7	9.9
Linear decay	37.0	10.5
Half Poisson	35.7	11.1
Half Gaussian	36.7	12.3

Table 6.3: Results of temporal action segmentation for different length regularizers on the Breakfast and the Hollywood Extended dataset. All results are based on a stop criterion of 5% frame change rate during alignment or a maximum of 15 iterations.

that $\max_{\ell > \text{len}(s_t)} \big\{p(\ell|s_t)\big\} = 1$, so that the constant part to the left of $\text{len}(s_t)$ and the right part with the Poisson decay are continuous.

Closely related to the half Poisson model is the **half Gaussian model**

$$p(\ell_t|s_t) = e^{-\frac{(\ell_t-\mu)^2}{\sigma^2}}. \tag{6.34}$$

Here, the property of a monotonically decreasing function is implicitly ensured by setting $\mu = 0$ and $\sigma = \text{len}(s_t)$.

We set a minimal probability $\varepsilon = 0.001$ for all length models. The models are illustrated in Figure 6.13.

We now evaluate the overall performance of the different functions on the system. For all measures, we use the stop criterion as described in Section 6.3.5. As Table 6.3 shows, the length regularizer mainly improves the results for the Breakfast dataset. All length models are doing better than the original system without length regularization in this case, with a best accuracy of 37.0% reached by the linear decay function. This is further supported by the evaluation of the performance during training as the plot in Figure 6.14 shows. Here, the solid line shows the segmentation accuracy of the models after each iteration until the stop criterion is reached. After that, additional results are displayed by a dashed line. It shows that all four functions significantly outperform the system without length regularization with best results at 36.7% for box and 35.7% and 36.7% for half Poisson and Gaussian.

The impact of the length models on the Hollywood Extended dataset is smaller than on the Breakfast dataset as shown in Table 6.3. This behavior can be based on the fact that the actions in the Hollywood Extended dataset are usually shorter and the action classes have a lower temporal variance compared to the Breakfast dataset. On Hollywood Extended, all action classes usually have a consistent mean frame length, whereas in case of Breakfast, the mean lengths of action classes significantly vary. Thus, the benefit of using length models increases with the heterogeneity of the target action classes. Therefore, it can be expected that a length model has less impact in this case than for datasets with high temporal variance among the action classes.

Breakfast	
Model	Frame Accuracy
OCDC (Bojanowski et al., 2014)*	8.9
HTK (Kuehne et al., 2017)	25.9
ECTC (Huang et al., 2016)	27.7
TCFPN (Ding and Xu, 2018)	**38.4**
GRU-RNN	33.3
GRU + length regularization	36.7

Hollywood Extended	
Model	Jacc (IoU)
HTK (Kuehne et al., 2017)	8.6
TCFPN (Ding and Xu, 2018)	**12.6**
GRU-RNN	11.9
GRU + length regularization	12.3

Table 6.4: Comparison of temporal action segmentation performance for GRU based weak learning with other approaches. For the Breakfast dataset, we report performance as frame accuracy, for Hollywood Extended, we measure the Jaccard index as intersection over union for this task (*from Huang et al. (2016)).

Overall, based on the numbers in Table 6.3, it can be stated that the half Gaussian function gives the most consistent improvement for both datasets. We therefore use length regularization with a half Gaussian function for the following experiments.

6.4.5 Comparison to State-of-the-Art

Temporal Action Segmentation. We compare our system to four different approaches published for this task: The first is the Ordered Constrained Discriminative Clustering (OCDC) proposed by Bojanowski et al. (2014), which has been introduced on the Hollywood Extended dataset. Second, we compare against the HTK system used by Kuehne et al. (2017), third against the Extended Connectionist Temporal Classification (ECTC) by Huang et al. (2016) and fourth against the temporal convolutional feature pyramid network (TCFPN) by Ding and Xu (2018).

Results are shown in Table 6.4. One can see that both GRU systems show a good performance, and that both systems are only slightly worse than TCFPN.

Temporal Action Alignment. We also address the task of action alignment. We assume that given a video and a sequence of temporally ordered actions, the task is to infer the

Breakfast	
Model	Jacc. (IoD)
OCDC (Bojanowski et al., 2014)	23.4
HTK (Kuehne et al., 2017)	42.4
TCFPN (Ding and Xu, 2018)	52.3
GRU-RNN	47.3
GRU + length regularizer	**52.4**

Hollywood Extended	
Model	Jacc. (IoD)
OCDC (Bojanowski et al., 2014)**	43.9
HTK (Kuehne et al., 2017)**	46.0
ECTC (Huang et al., 2016)**	41.0
TCFPN (Ding and Xu, 2018)	39.6
GRU-RNN	**46.3**
GRU + length regularizer	46.0

Table 6.5: Results for temporal action alignment on the test set of the Breakfast and the Hollywood Extended dataset reported as Jaccard index of intersection over detection (IoD)(**results obtained from the authors).

respective boundaries for the given action order. We report results for the test set of Breakfast as well as for the Hollywood Extended dataset based on the Jaccard index (Jacc.) computed as intersection over detection (IoD) as proposed by Bojanowski et al. (2014). The results are shown in Table 6.5.

The proposed approach outperforms current state-of-the-art approaches. It also shows that the system without length model performs slightly better than the system with length model for the alignment in case of Hollywood Extended. As already discussed in Section 6.4.4, the differences between the proposed approach with and without length model are marginal on this dataset.

Fully supervised classification. We finally evaluate the approach in a fully supervised setting and compare it to other proposed approaches. Here, we compute the mean length from the training annotations directly and use it to determine the number of states as well as the length model parameters of the regularizer function.

Table 6.6 shows that the system clearly outperforms previous approaches. Since the approaches from Kuehne et al. (2014) and Kuehne et al. (2016) have a similar hierarchical structure as the presented system and similar features, we can assume that the main improvement can be attributed to the underlying GRU models. This is consistent with the findings

Breakfast	
Model	Frame Accuracy
HMM-BOW (Kuehne et al., 2014)	28.8
HMM-FV (Kuehne et al., 2016)	56.3
TCFPN (Ding and Xu, 2018)	52.0
GRU w/o length regularizer	**60.2**
GRU + length regularizer	**61.3**

Table 6.6: Results for fully supervised temporal action segmentation on the Breakfast dataset.

in Section 6.4.2, where it shows that the proposed GRUs improve fine-grained frame-based classification compared to other approaches. Additionally, it shows that the length model also improves the segmentation accuracy in case of fully supervised training. This is important as in this case, we can assume that all other temporal factors, such as the number of states are already optimal. Thus, even in this case, a temporal regularizer can improve the overall recognition of the system.

6.5 Summary

We presented an approach for weakly supervised learning of human actions based on a combination of a discriminative representation of subactions modeled by a recurrent neural network and a coarse probabilistic model to allow for a temporal alignment and inference over long sequences. Although the system itself shows already good results, the performance is significantly improved by approximating the number of subactions for the different action classes and by adding a length regularization to the overall system. Accordingly, we combine the length model with the adaptation of the number of subaction classes by iterating realignment and reestimation during training. The resulting model shows a competitive performance on various weak learning tasks such as temporal action segmentation and action alignment on two standard benchmark datasets.

Chapter 7

NeuralNetwork-Viterbi

The possibility to train a system for temporal action segmentation in a weakly supervised fashion is a huge step towards being able to process vast amounts of video data without extensive annotation. Current methods, including our approach discussed in Chapter 6, however, are still of poor quality. With a frame accuracy of only 33% on Breakfast, for instance, the approach presented in the previous chapter is of limited use for practical applications and there is need for stronger models that allow for higher accuracies.

In this chapter, we address the main weaknesses of the previously proposed method and work towards an end-to-end trainable architecture by integrating the Viterbi re-alignment directly into the loss computation for the visual model. We also close the gap between weak supervision based on action transcripts and full supervision based on full frame annotations by introducing a semi-supervised setup, where only a few frames per video are manually annotated.

Contents

7.1 Introduction

While the method proposed in the previous chapter is a first step towards handling the continuously growing amount of publicly available video data on YouTube or video streaming services, its performance is not yet good enough for practical applications such as surveillance or the analysis of continuous video streams e.g. in the domain of autonomous driving.

In the context of classical action recognition with pre-segmented clips, the best performing approaches are deep neural networks that can be trained in an end-to-end fashion, see e.g. Simonyan and Zisserman (2014); Wang et al. (2016); Carreira and Zisserman (2017); Feichtenhofer et al. (2017a). End-to-end learning is less straightforward for applications that require the segmentation of temporally untrimmed videos which usually contain a large variety of different actions with different lengths as current architectures like the one proposed in the previous chapter usually contain non-differentiable iterative steps and operations such as an $\arg\max$.

In this chapter, we revisit the task of temporal action segmentation with action transcripts as weak supervision discussed in the previous chapter. In order to learn a model for temporal action segmentation with such weak supervision, so far we relied on a fine-to-coarse model with an RNN as visual model, an explicit HMM for the intra-class temporal progression, and the grammar as a model for inter-class context.

That approach, similar to related HMM-based methods by Koller et al. (2016); Kuehne et al. (2017) and Koller et al. (2017), comes with the major problem that its training requires some heuristic ground truth. HMM-based approaches typically rely on a two-step approach that is iterated several times. It consists of first generating a segmentation for each training video using the Viterbi algorithm and then training the neural network as in the fully supervised case using the generated segmentation as pseudo ground truth. Consequently, the two-step approach is sensitive to the initialization of the pseudo ground truth and the accuracy tends to oscillate between the iterations. Avoiding a two-step scheme, ECTC (Graves et al., 2006) provides a framework for weakly supervised sequence learning. However, this approach does not allow to include explicit models for the context between classes and their temporal progression and therefore does not achieve state-of-the-art performance.

To overcome the aforementioned issues, we propose a novel learning algorithm that allows for direct learning using the input video and ordered action transcript only. The approach includes the Viterbi-decoding as part of the loss function to train the neural network and has several practical advantages compared to the two-stage approach: it neither suffers from an oscillation effect nor requires a frame-wise labeling as initialization or any kind of pseudo ground truth, models are learned incrementally, and the accuracy is improved due to direct optimization of the loss function of interest.

As a second contribution, we also propose to use an explicit length model instead of the HMMs used in the previous chapter, allowing to learn the action classes directly rather than intermediate HMM states.

The remainder of this chapter is structured as follows. In Section 7.2, we discuss the drawbacks of pseudo ground truth based methods and introduce a novel, direct learning scheme. In Section 7.3, the proposed method is evaluated and empirically compared to the previously proposed approach from Chapter 6. Finally, in Section 7.4, we also apply the model to semi-supervised learning where frames are annotated sparsely.

7.2 Technical Details

In this section, we build upon the model introduced in Chapter 6 and simplify it such that the HMM is obsolete. Therefore, we start from the same generic framework the preceding approaches also built upon. Considering the general model that is introduced in Section 4.2,

$$\begin{aligned}(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) &= \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{c}_1^N, \mathbf{l}_1^N | \mathbf{x}_1^T) \Big\} \\ &= \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) \cdot p(\mathbf{l}_1^N | \mathbf{c}_1^N) \cdot p(\mathbf{c}_1^N) \Big\}, \end{aligned} \tag{7.1}$$

we discuss drawbacks in the training procedure of the approach introduced in Chapter 6.

The HMM-based approach, together with several state-of-the-art methods (Vo and Bobick, 2014; Kuehne et al., 2017; Koller et al., 2016, 2017), formulates $p(\mathbf{c}_1^N, \mathbf{l}_1^N | \mathbf{x}_1^T)$ in such a way such that the arg max can be efficiently computed using a Viterbi-like algorithm at the cost of having three model components – the visual model, the length model, and the context model – that are typically trained separately. In fully supervised settings such as the one presented in Chapter 5 or the approach of Vo and Bobick (2014) frame-wise labeling of the training videos allows for a reliable and consistent training of the three model components although they are not trained jointly.

In a weakly supervised setting like in the works of Kuehne et al. (2017); Koller et al. (2016, 2017), or the approach introduced in Chapter 6, a frame-level ground truth is not available during training and inconsistencies between the separately trained models can have a larger effect.

In this Chapter, we revisit the problem of weakly supervised learning and propose two improvements. The first one addresses the modeling of $p(\mathbf{c}_1^N, \mathbf{l}_1^N | \mathbf{x}_1^T)$. Instead of using a hidden Markov model as in Kuehne et al. (2017); Koller et al. (2016, 2017), or Chapter 6, we explicitly model the length of each action class. The model is described in Section 7.2.3 and in our experiments we show that the proposed length model outperforms an HMM. The second novelty is a more principled approach for weakly supervised learning. This approach is described in Section 7.2.1 and can be used to train any model that uses neural networks and Viterbi decoding.

Before we describe the proposed learning approach, we briefly discuss the connection between training in a fully supervised setting and the training procedure that is used in Chapter 6 for weakly supervised learning.

In a classical fully supervised training setup, frame-wise ground truth annotation is provided for the training data, i.e. each training video can be regarded as a triple $(\mathbf{x}_1^T, \mathbf{c}_1^N, \mathbf{l}_1^N)$. Since $\mathbf{l}_1^N$ and therefore the label $c_{n(t)}$ for each frame x_t is known, the underlying visual model for Equation (7.1), which is typically a neural network, is trained using the frame-level annotations and, for instance, the cross-entropy loss.

If only the transcript of a training video, i.e. an ordered sequence of classes that occur in the video, is given, $\mathbf{l}_1^N$ is unknown and only $(\mathbf{x}_1^T, \mathbf{c}_1^N)$ is provided. In the previously discussed HMM-RNN approach, the training is basically reduced to the problem of fully supervised training by generating a pseudo ground truth $c_{n(t)}^{\text{pseudo}}$ for all training sequences based on either the uniform alignment at the first training iteration or re-alignments in later iterations. A neural network is then trained using a pseudo cross-entropy loss that is based on the pseudo ground truth $c_{n(t)}^{\text{pseudo}}$.

This approach comes with a major problem: The actual learning phase and the transcript decoding (i.e. pseudo ground truth generation) are separated and the transcripts $\mathbf{c}_1^N$ are only used for the pseudo ground truth generation. In other words, the model learning does not explicitly include the transcripts.

As a workaround, the two steps *pseudo ground truth generation* and *model learning* are repeated several times, where the pseudo ground truth in the first iteration is a uniform alignment of transcripts to sequence frames. In later repetitions, the pseudo ground truth is generated using a Viterbi decoding on Equation (7.1) with the previously trained network. From a practical point, this results in several major limitations. As Figure 6.14 reveals, the approach is sensitive to the initialization of the pseudo ground truth and the accuracy tends to oscillate between the iterations. Furthermore, the approach processes the entire dataset in each step, which prevents its use for incremental learning.

7.2.1 NeuralNetwork-Viterbi

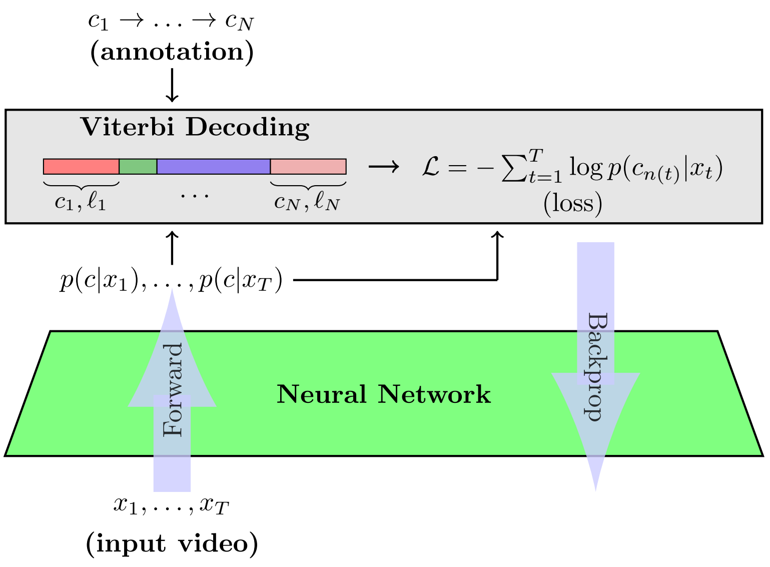
To overcome these issues, we propose a new framework that allows to learn directly from the transcripts. Therefore, we define a loss that can be computed solely based on the current model and a single training example $(\mathbf{x}_1^N, \mathbf{c}_1^N)$. The loss is designed to be zero if

$$p(\mathbf{c}_1^N, \mathbf{l}_1^N | \mathbf{x}_1^T) = p(\mathbf{c}_1^N, \mathbf{l}_1^N | \mathbf{x}_1^T, \mathbf{c}_1^N), \tag{7.2}$$

i.e. if the prediction without given transcripts (left hand side) is equal to the prediction with given transcripts (right hand side). Particularly, our approach does not require a precomputed pseudo ground truth and works directly on the weakly annotated data.

Our new training procedure is illustrated in Figure 7.1. The training algorithm randomly draws a sequence $\mathbf{x}_1^T$ and its annotation $\mathbf{c}_1^N$ from the training set. The sequence is then forwarded through a neural network. Note that there are no constraints on the network architecture, all commonly used feed-forward networks, CNNs, and recurrent networks can be used. The optimal segmentation by means of Equation (7.1) is then computed by application of a Viterbi decoding on the network output, see Section 7.2.4 for details. Since $\mathbf{c}_1^N$ is provided as annotation, only $\mathbf{l}_1^N$ needs to be inferred during training. We switch notation and write the Viterbi segmentation $(N, \mathbf{c}_1^N, \mathbf{l}_1^N)$ as framewise labels $c_{n(1)}, \dots, c_{n(T)}$, with which the cross-entropy loss over all aligned frames is accumulated:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p(c_{n(t)} | x_t). \tag{7.3}$$



$c_1 \rightarrow \ldots \rightarrow c_N$
(annotation)
Viterbi Decoding
c_1, ℓ_1
$\ldots$
c_N, ℓ_N
$\longrightarrow \mathcal{L} = -\sum_{t=1}^{T} \log p(c_{n(t)}|x_t)$
(loss)
$p(c|x_1), \ldots, p(c|x_T)$
Forward
Neural Network
Backprop
$x_1, \ldots, x_T$
(input video)

Figure 7.1: The input video $\mathbf{x}_1^T$ is forwarded through the network and the Viterbi decoding is run on the output probabilities. The frame labels generated by the Viterbi algorithm are then used to compute a framewise cross-entropy loss based on which the network gradient is computed.

We chose the cross-entropy loss as it is most common in neural network optimization. However, our framework is not bound to a specific loss function. Once the Viterbi segmentation of the input sequence is computed, any other loss such as squared-error can as well be used.

Based on the sequence loss $\mathcal{L}$, the network parameters are updated using stochastic gradient descent with the gradient $\nabla\mathcal{L}$ of the loss. We would like to emphasize that the algorithm operates in an online fashion, i.e. in each iteration, the loss $\mathcal{L}$ is computed with respect to a single randomly drawn training sequence $(\mathbf{x}_1^T, \mathbf{c}_1^N)$ only.

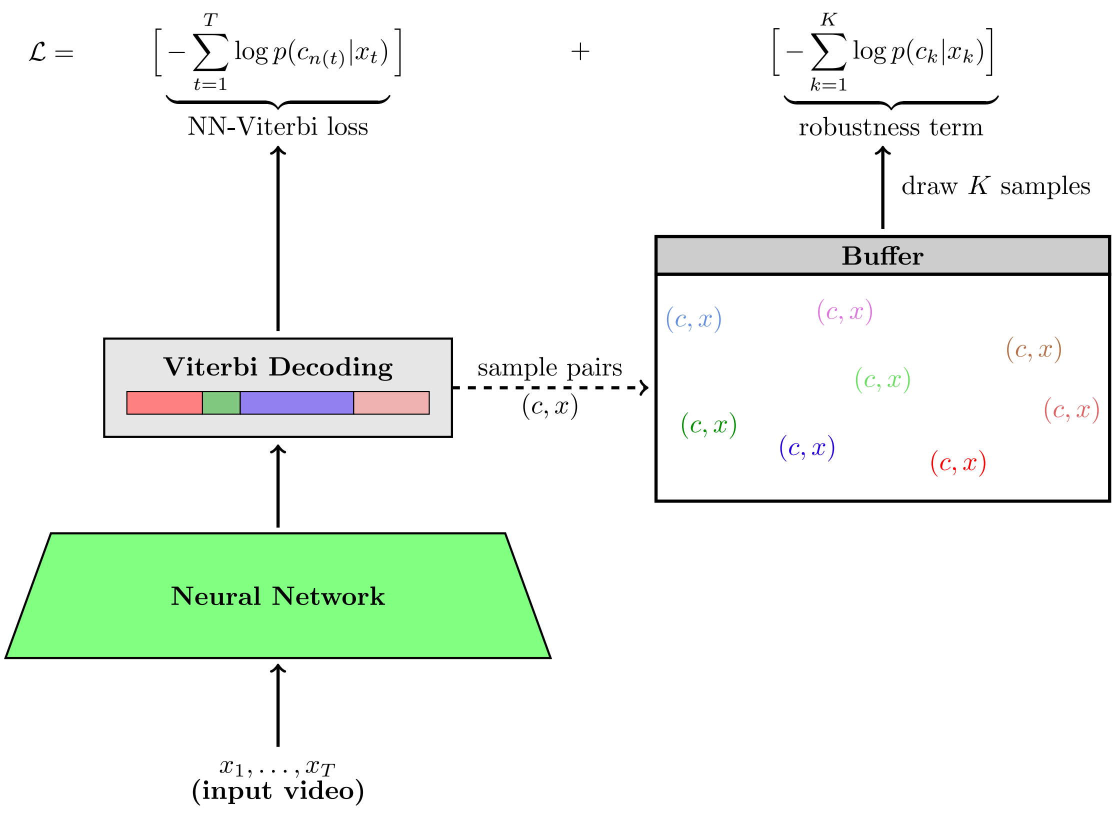
7.2.2 Enhancing the Robustness

In practice, a sequence $\mathbf{x}_1^T$ can easily be a few thousand frames long. Backpropagating all frames at once can thus raise problems with the limited GPU memory. Moreover, online learning algorithms generally benefit from making a large number of model updates. Therefore, we split the sequence into multiple mini-batches after the Viterbi segmentation $c_{n(1)}, \ldots, c_{n(T)}$ has been computed. These minibatches are then backpropagated one-by-one through the network.

However, traditional online learning algorithms such as stochastic gradient descent rely on the assumption that

$$\mathcal{L}^*(w) = \mathbb{E}_{\mathbf{x}}\mathcal{L}(x, w) = \int \mathcal{L}(x, w) d\mathcal{P}(x), \tag{7.4}$$

where w denotes the model parameters, $\mathcal{L}^*(w)$ is the true loss that is to be optimized, and $\mathcal{L}(x, w)$ is the loss of a single observation x, see e.g. Bottou (1998). In each iteration, the



$\mathcal{L} = \underbrace{\Big[-\sum_{t=1}^{T}\log p(c_{n(t)}|x_t)\Big]}_{\text{NN-Viterbi loss}} + \underbrace{\Big[-\sum_{k=1}^{K}\log p(c_k|x_k)\Big]}_{\text{robustness term}}$
draw K samples
Buffer
(c, x)
Viterbi Decoding
sample pairs
(c, x)
Neural Network
$x_1, \dots, x_T$
(input video)

Figure 7.2: Illustration of the robustness enhancement. After each Viterbi alignment, frame/label pairs are sampled and stored in the buffer that already contains frame/label pairs from previous iterations. The loss is then a sum of the cross-entropy loss on the currently decoded sequence (NN-Viterbi loss) and the framewise cross-entropy of samples drawn from the buffer (robustness term).

observations x are usually assumed to be drawn independently from the distribution $\mathcal{P}(x)$. In our setting, on the contrary, all frames in an iteration belong to the same sequence $\mathbf{x}_1^T$, so they are not independent. Further subdividing long sequences into smaller mini-batches enhances the problem: multiple updates are made with a strong bias towards *(a)* the characteristics of the sequence frames and *(b)* the limited amount of classes occurring in the sequence.

We therefore propose to use a buffer $\mathcal{B}$ and store recently processed sequences and their inferred frame labels. In order to make the gradient in each iteration more robust, K frames from the buffer are sampled and added to the loss function,

$$\mathcal{L} = -\Big[\sum_{t=1}^{T}\log p(c_{n(t)}|x_t) + \sum_{k=1}^{K}\log p(c_k|x_k)\Big]. \tag{7.5}$$

The process is illustrated in Figure 7.2. Since the neural network is updated gradually in small steps, most of the frame/label pairs in the buffer still agree with the current model. However, sampling random frames from the buffer lessens the above-mentioned sequence bias from the loss function and increases the robustness of the optimization algorithm.

7.2.3 The Model

We now introduce the specific model used in this Chapter. Using the usual starting point from Equation 4.2,

$$\begin{aligned}(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) &= \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{c}_1^N, \mathbf{l}_1^N | \mathbf{x}_1^T) \Big\} \\ &= \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) \cdot p(\mathbf{l}_1^N | \mathbf{c}_1^N) \cdot p(\mathbf{c}_1^N) \Big\}, \end{aligned} \tag{7.6}$$

we define the three components.

For the visual model, we use the factorization

$$p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) = \prod_{t=1}^{T} p(x_t | \mathbf{x}_1^{t-1}, \mathbf{c}_1^N, \mathbf{l}_1^N). \tag{7.7}$$

Dropping the dependence on the length and assuming conditional independence of the frames and that they depend only on the class of the current segment, the visual model further simplifies to

$$p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) = \prod_{t=1}^{T} p(x_t | c_{n(t)}). \tag{7.8}$$

Similar to the previous approach, we use a neural network to model the visual probabilities $p(x_t|c_{n(t)})$. We use a recurrent network with a single layer of 64 gated recurrent units and a softmax output. A similar architecture has also been used in the previous chapter with the only difference that the output of the network were HMM states and now the GRU predicts action classes directly. The method of Huang et al. (2016) also uses a similar network architecture. In contrast to previous approaches, however, we train the network with the NeuralNetwork-Viterbi loss as described in Section 7.2.1. Since the outputs of the neural network are posterior probabilities $p(c|x_t)$, we follow the hybrid approach (Bourlard and Morgan, 2012) and refactor

$$p(x_t|c) \propto \frac{p(c|x_t)}{p(c)}, \tag{7.9}$$

where $p(c)$ is a class prior that also needs to be learned during training. Therefore, we use a running estimate by counting the amount of frames that have been labeled with a class c for all sequences that have been processed so far. Normalizing these counts to sum up to one finally results in the estimate of $p(c)$. The prior is updated after every iteration, i.e. after every new training sequence. If a sequence annotation $\mathbf{c}_1^N$ contains a class that has not been seen before, $1/|\mathcal{C}|$ is used.

For the length model, we assume that lengths of different segments are independent of each other and only depend on the class label of the same segment,

$$p(\mathbf{l}_1^N|\mathbf{c}_1^N) = \prod_{n=1}^{N} p(\ell_n|c_n), \tag{7.10}$$

and we model the class-conditional length distribution with a class-dependent Poisson model,

$$p(\ell|c) = \frac{\lambda_c^\ell}{\ell!} e^{-\lambda_c}. \tag{7.11}$$

For training, we again rely on a running estimate. After each iteration, λ_c is updated, which is the mean length of a segment for class c. If the training sample $(\mathbf{x}_1^T, \mathbf{c}_1^N)$ contains a class that has not been seen before, we set $\lambda_c = N/T$. Note that such running estimates are also used in other neural network layers such as batch-normalization (Ioffe and Szegedy, 2015).

The context model is the same as proposed in Section 6.3.2, i.e. a right-regular grammar that produces exactly the action transcripts that occur in the training set. Similar to the HMM-based approach in Chapter 7, during training we ensure that the Viterbi decoding generates a segmentation that matches the given action transcript by using a grammar that can produce this transcript only.

Inserting these models into Equation 7.6, the overall model is

$$(\hat{N}, \mathbf{c}_1^{\hat{N}}, \mathbf{l}_1^{\hat{N}}) = \operatorname*{arg\,max}_{\substack{N, \mathbf{c}_1^N, \mathbf{l}_1^N, \\ \exists \mathbf{h}_1^N:\ h_{n-1} \to c_n\ h_n \in \mathcal{R}}} \Big\{ \prod_{t=1}^{T} p(x_t|c_{n(t)}) \cdot \prod_{n=1}^{N} p(\ell_n|c_n) \cdot p(c_n|h_{n-1}) \Big\}. \tag{7.12}$$

Note that we follow a similar trick as for the HMM based model to make sure the arg max in only computed over action sequences $\mathbf{c}_1^N$ that can be produced by the context model, a constraint is added that requires the existence of a sequence $\mathbf{h}_1^N$ of non-terminal symbols that allows a decomposition into actions $\mathbf{c}_1^N$.

7.2.4 Inference

In order to solve the maximization in (7.12), we again rely on a variation of the Viterbi algorithm. As outlined above, this algorithm is not only used for inference as in the fully supervised case described in Chapter 5 but also during training.

Using the function $n(t)$ from Equation (4.5) that maps from timeframes to segment indices and the index function $[\![\cdot]\!]$, Equation (7.12) is rewritten as a product over all frames,

$$(\hat{N}, \mathbf{c}_1^{\hat{N}}, \mathbf{l}_1^{\hat{N}}) = \operatorname*{arg\,max}_{\substack{N, \mathbf{c}_1^N, \mathbf{l}_1^N, \\ \exists \mathbf{h}_1^N:\ h_{n-1} \to c_n h_n \in \mathcal{R}}} \Big\{ \prod_{t=1}^{T} p(x_t|c_{n(t)}) \cdot \big[p(\ell_{n(t)}|c_{n(t)}) \cdot p(c_{n(t)}|h_{n(t)-1})\big]^{[\![n(t) \neq n(t-1)]\!]} \Big\}, \tag{7.13}$$

where the index function evaluates to one if the segment index at time t is different from the segment index at time $t-1$, i.e. if a new segment starts at time t. In this formulation, both context model and length model are multiplied already at the first frame of the respective segment. For the dynamic programming recursions, which run sequentially over the time axis, this requires a look-ahead to the length of a segment when its first frame is hypothesized. Since this would result in an inefficient algorithm, we modify Equation (7.13) such that the length model for segment n is multiplied at the first frame of segment $n+1$,

$$(\hat{N}, \mathbf{c}_1^{\hat{N}}, \mathbf{l}_1^{\hat{N}}) = \underset{\substack{N, \mathbf{c}_1^N, \mathbf{l}_1^N, \\ \exists \mathbf{h}_1^N:\ h_{n-1} \to c_n h_n \in \mathcal{R}}}{\arg\max} \Big\{ p(\ell_{n(T)}|c_{n(T)}) \cdot \prod_{t=1}^{T} p(x_t|c_{n(t)}) \cdot \big[p(\ell_{n(t)-1}|c_{n(t)-1}) \cdot p(c_{n(t)}|h_{n(t)-1})\big]^{[\![n(t) \neq n(t-1)]\!]} \Big\}. \quad (7.14)$$

Note that we assume $n(1) \neq n(0)$ evaluates to true and that $p(\ell_0|c_0) = 1$ for the undefined case of $n(t)-1 = 0$. Moreover, the length model for the last segment is multiplied separately, so that Equation (7.14) is equivalent to Equation (7.13).

Similar to the previous derivations of the recursive equations, we define an auxiliary function $Q(\tau, \ell, c, h)$ that yields the best probability score for a segmentation up to frame τ meeting the following conditions:

1. the latest segment is ℓ frames long at time τ (but does not necessarily end here),
2. the class label of the latest segment is c,
3. the nonterminal symbol of the stochastic grammar is h,

such that the optimal segmentation score in terms of Equation (7.14) is

$$\max_{\ell,c} \Big\{ p(\ell|c) \cdot Q(T, \ell, c, h_{\text{end}}) \Big\} \quad (7.15)$$

with h_{end} denoting a virtual end symbol of the grammar, as introduced in Section 6.3.3. A comparison of Equation (7.14) and Equation (7.15) reveals the exact definition of the auxiliary function Q,

$$Q(\tau, \ell, c, h) = \max_{\substack{\eta, \mathbf{c}_1^\eta, \mathbf{l}_1^\eta, \\ \exists \mathbf{h}_1^\eta:\ h_{n-1} \to c_n\ h_n \in \mathcal{R}, \\ \ell_\eta = \ell, c_\eta = c, h_\eta = h}} \Big\{ \prod_{t=1}^{\tau} p(x_t|c_{n(t)}) \cdot \big[p(\ell_{n(t)-1}|c_{n(t)-1}) \cdot p(c_{n(t)}|h_{n(t)-1})\big]^{[\![n(t) \neq n(t-1)]\!]} \Big\}, \quad (7.16)$$

where the condition on the last segment label c_η, the last segment length ℓ_η, and the last context h_η ensures that the segmentation $(\eta, \mathbf{c}_1^\eta, \mathbf{l}_1^\eta)$ ends in (ℓ, c, h) as required by Q.

We again separate the last factor at time τ in a case distinction between the within segment case and the between segment case.

For the within segment case, the segment index does not change between $\tau - 1$ and τ and we have

$$\begin{aligned} Q(\tau, \ell, c, h) = &\max_{\substack{\eta, \mathbf{c}_1^\eta, \mathbf{l}_1^\eta, \\ \exists \mathbf{h}_1^\eta:\ h_{n-1} \to c_n\ h_n \in \mathcal{R}, \\ \ell_\eta = \ell, c_\eta = c, h_\eta = h}} \Big\{ \prod_{t=1}^{\tau-1} p(x_t|c_{n(t)}) \cdot \big[p(\ell_{n(t)-1}|c_{n(t)-1}) \cdot p(c_{n(t)}|h_{n(t)-1})\big]^{[\![n(t) \neq n(t-1)]\!]} \Big\} \cdot p(x_\tau|c) \\ = &\ Q(\tau - 1, \ell - 1, c, h) \cdot p(x_\tau|c). \end{aligned} \quad (7.17)$$

For the between segment case, it holds that $n(\tau) \neq n(\tau-1)$ and, as a new segment is hypothesized, $\ell = 1$. In this case, isolating the last frame τ requires to isolate the respective length and context models, too,

$$Q(\tau, \ell=1, c, h) = \max_{\substack{\eta, \mathbf{c}_1^\eta, \mathbf{l}_1^\eta, \\ \exists \mathbf{h}_1^\eta:\ h_{n-1} \to c_n\ h_n \in \mathcal{R}, \\ \ell_\eta = \ell, c_\eta = c, h_\eta = h}} \Big\{ \Big[\prod_{t=1}^{\tau-1} p(x_t|c_{n(t)}) \cdot \big[p(\ell_{n(t)-1}|c_{n(t)-1}) \cdot p(c_{n(t)}|h_{n(t)-1})\big]^{[\![n(t)\neq n(t-1)]\!]} \Big] \cdot p(x_\tau|c_\eta) \cdot p(\ell_{\eta-1}|c_{\eta-1}) \cdot p(c_\eta|h_{\eta-1}) \Big\}. \tag{7.18}$$

Since the maximization over the isolated factors of the last segment does not affect the product over the first $\tau-1$ frames, the maximization over the last segment can be separated in order to obtain the recursive equation,

$$\begin{aligned} Q(\tau, \ell=1, c, h) &= \max_{\substack{\tilde{\ell}, \tilde{c}, \tilde{h}: \\ \tilde{h} \to c\ h \in \mathcal{R}, \\ \exists h' \to \tilde{c}\ \tilde{h}\ \in \mathcal{R}}} \Big\{ \max_{\substack{\tilde{\eta}, \mathbf{c}_1^{\tilde{\eta}}, \mathbf{l}_1^{\tilde{\eta}}, \\ \exists \mathbf{h}_1^{\tilde{\eta}}:\ h_{n-1} \to c_n\ h_n \in \mathcal{R}, \\ \ell_{\tilde{\eta}} = \tilde{\ell}, c_{\tilde{\eta}} = \tilde{c}, h_{\tilde{\eta}} = \tilde{h}}} \Big[\prod_{t=1}^{\tau-1} p(x_t|c_{n(t)}) \cdot \big[p(\ell_{n(t)-1}|c_{n(t)-1}) \cdot p(c_{n(t)}|h_{n(t)-1})\big]^{[\![n(t)\neq n(t-1)]\!]} \Big] \cdot p(x_\tau|c) \cdot p(\tilde{\ell}|\tilde{c}) \cdot p(c|\tilde{h}) \Big\} \\ &= \max_{\substack{\tilde{\ell}, \tilde{c}, \tilde{h}: \\ \tilde{h} \to c\ h \in \mathcal{R}, \\ \exists h' \to \tilde{c}\ \tilde{h}\ \in \mathcal{R}}} \big\{ Q(\tau-1, \tilde{\ell}, \tilde{c}, \tilde{h}) \cdot p(x_\tau|c) \cdot p(\tilde{\ell}|\tilde{c}) \cdot p(c|\tilde{h}) \big\}. \end{aligned} \tag{7.19}$$

In order to reconstruct the best segmentation, traceback arrays are used similar to the previous versions of Viterbi-style algorithms. Since Q optimizes not only over the action classes and contexts, the length is also part of the traceback. For compactness of notation, we store the required traceback triplet $(\tilde{\ell}, \tilde{c}, \tilde{h})$ in a single array and define

$$A(\tau, \ell, c, h) = (\ell, c, h) \tag{7.20}$$

for the within segment case and

$$A(\tau, \ell, c, h) = \operatorname*{arg\,max}_{\substack{\tilde{\ell}, \tilde{c}, \tilde{h}: \\ \tilde{h} \to c\ h \in \mathcal{R}, \\ \exists h' \to \tilde{c}\ \tilde{h}\ \in \mathcal{R}}} \big\{ Q(\tau-1, \tilde{\ell}, \tilde{c}, \tilde{h}) \cdot p(x_\tau|c) \cdot p(\tilde{\ell}|\tilde{c}) \cdot p(c|\tilde{h}) \big\} \tag{7.21}$$

for the between segment case. Both cases, within segment and between segment, are illustrated in Figure 7.3 and Figure 7.4.

Complexity

Implementations of the Viterbi decoding and the traceback are provided as pseudo code in Algorithm 7.1 and Algorithm 7.2.

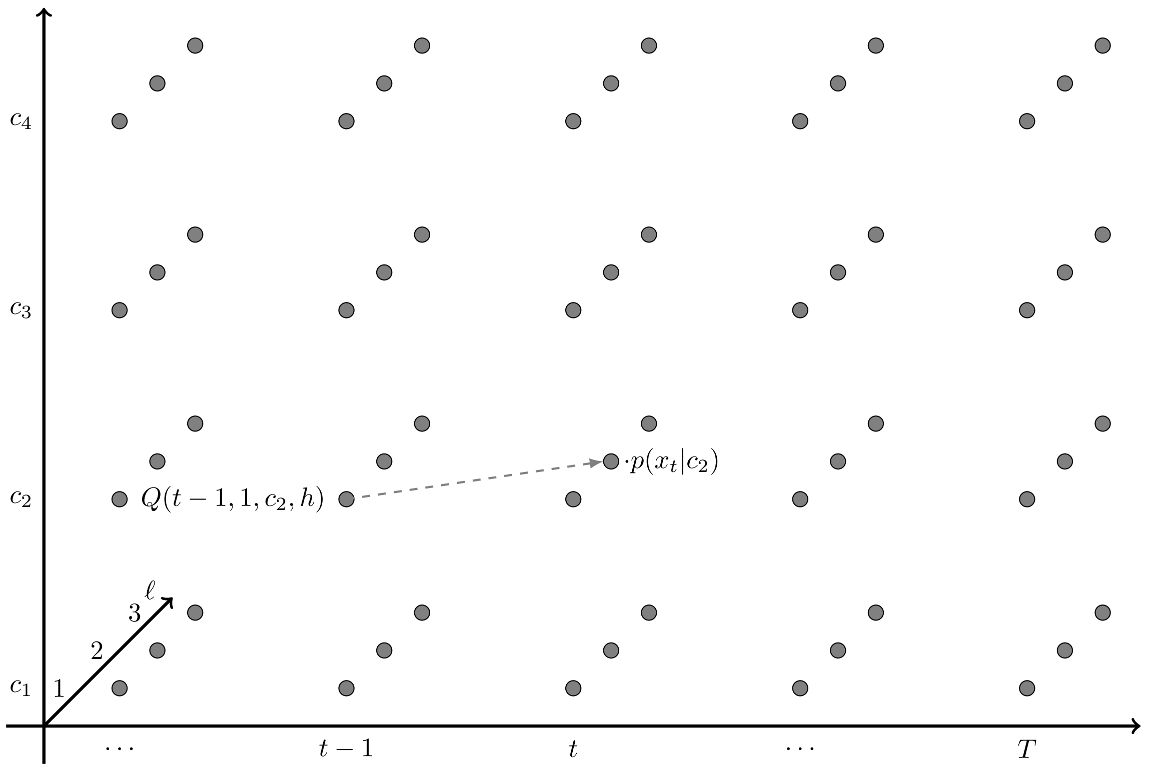


Figure 7.3: Viterbi decoding for the within segment case. The new score is the score of the hypothesis at time $t-1$ with length $\ell-1$ that has the same class and context multiplied by the probability of the visual model. In contrast to previous approaches, there is an additional axis for the length to optimize over.

The Viterbi decoding is, compared to the algorithm from the previous chapter, explicitly depended on the lengths of action segments. In order to fill the scores and tracebacks $Q(t, \ell, c, h)$ and $A(t, \ell, c, h)$, the algorithm needs to run over all video frames and evaluate at each frame each possible length and each possible class. The context h does not need to run over all possible non-terminal symbols but only needs to account for those symbols that occur together with class c on the right-hand-side of a rule. For the within segment case ($\ell > 1$), the situation is simple: all $Q(t, \ell > 1, c, h)$ values can be computed in $\mathcal{O}(T \cdot L \cdot |\mathcal{C}| \cdot |\mathcal{R}|)$, where L denotes the maximal action length.

For the between segment case ($\ell = 1$), the length of the current segment is fixed to one but a maximization over all predecessor lengths is required, see Equation (7.19). Moreover, there is a nested for-loop over the rules of the grammar (lines five, seven, and eight of Algorithm 7.1). This can lead to a runtime being cubic in the number of rules in principle. The kind of grammar we use, however, leads to a runtime linear in the number of rules: since the grammar we use generates exactly the transcripts that occur in the training set, it has a tree structure as illustrated in Figure 6.5, so that in line seven and eight of Algorithm 7.1, only a single matching rule exists. The overall runtime of the Viterbi decoding is thus

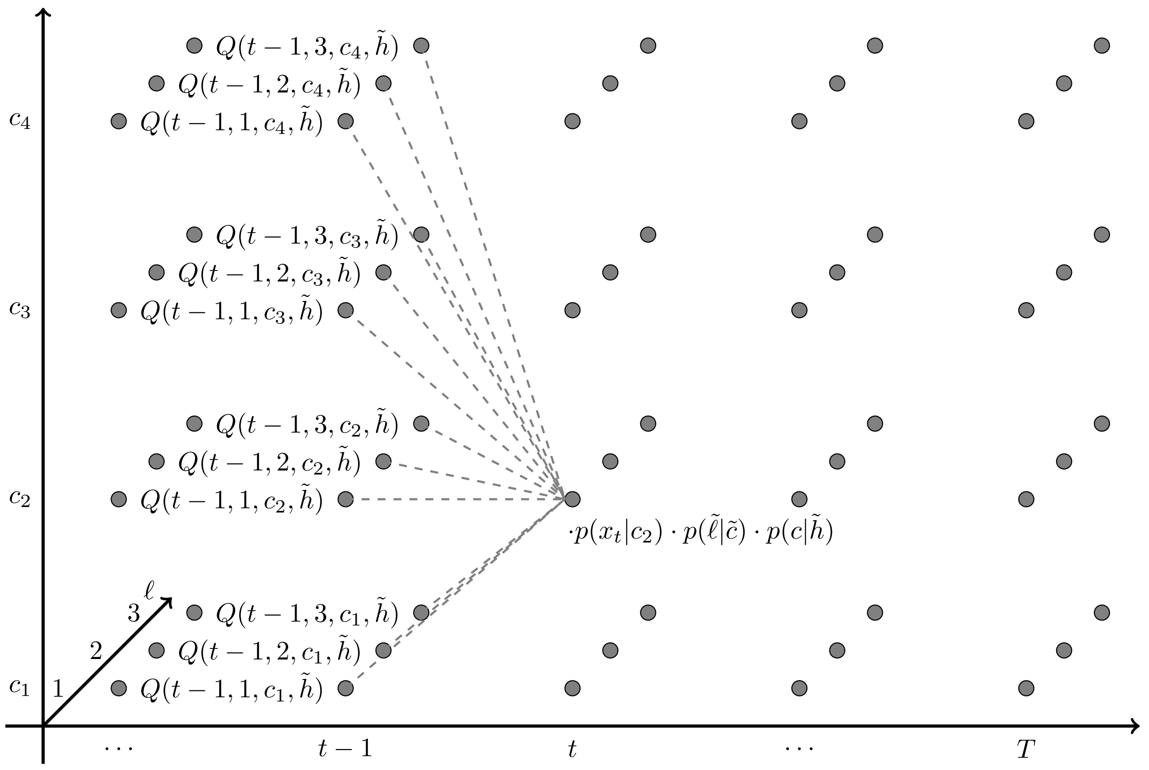


Figure 7.4: Viterbi decoding for the between segment case. In order to hypothesize a new segment in frame t, the hypotheses at frame $t-1$ have to be taken into account for any length and class. Note that we omitted context symbol axis for visualization purposes.

$\mathcal{O}(T \cdot L \cdot |\mathcal{C}| \cdot |\mathcal{R}|)$. In practice, pruning of bad hypotheses can significantly speed up the decoding.

The reconstruction of the best segmentation (Algorithm 7.2) consists of two steps. The first is finding the best ending hypothesis at time T, which requires a search over all possible lengths and classes. The second step is a backward pass through time to track the segment start positions. New segment beginnings can be found by analyzing if the length keeps decreasing. The runtime of the traceback is thus $\mathcal{O}\big(\max(T, L \cdot |\mathcal{C}|)\big)$.

7.3 Experiments

In the following, we analyze the components of the proposed NeuralNetwork-Viterbi algorithm and evaluate the effect of explicit length modeling compared to implicit length modeling by HMMs. Moreover, we show that the direct loss that immediately incorporates the action transcripts without the need for an explicit pseudo ground truth also allows for incremental learning. A comparison to existing approaches finally shows huge improvements in accuracy. We conclude the section with a discussion about semi-supervised learning, where sparse frame labels are available.

Algorithm 7.1 Viterbi Decoding

```
1:  $Q(1:T,:,:,:) = 0$
2:  $Q(0,:,:,h_\$) = 1$
3:  for $t = 1,\dots,T$ do
4:      for $c \in \mathcal{C}$ do
5:          for $h : \exists \tilde{h} \rightarrow c\ h \in \mathcal{R}$ do
6:              for $\ell = 1,\dots,L$ do
7:                  for $\tilde{h} : \tilde{h} \rightarrow c\ h \in \mathcal{R}$ do
8:                      for $\tilde{c} : \exists h' \rightarrow \tilde{c}\ \tilde{h}\ \in \mathcal{R}$ do
9:                          if $Q(t,1,c,h) < Q(t-1,\ell,\tilde{c},\tilde{h}) \cdot p(x_t|c) \cdot p(\ell|\tilde{c}) \cdot p(c|\tilde{h})$ then
10:                             $Q(t,1,c,h) = Q(t-1,\ell,\tilde{c},\tilde{h}) \cdot p(x_t|c) \cdot p(\ell|\tilde{c}) \cdot p(c|\tilde{h})$
11:                             $A(t,1,c,h) = (\ell,\tilde{c},\tilde{h})$
12:                 if $\ell > 1$ then
13:                     $Q(t,\ell,c,h) = Q(t-1,\ell-1,c,h) \cdot p(x_t|c)$
14:                     $A(t,\ell,c,h) = (\ell-1,c,h)$
15: return $Q, A$
```

7.3.1 Setup

We provide an in-depth analysis of the proposed method on the Breakfast dataset. For a comparison to other state-of-the-art approaches in action segmentation, we provide results on Breakfast and 50 Salads. For the task of action alignment, we provide an evaluation on Hollywood Extended. Details on the datasets can be found in Chapter 2.

In accordance with the approach presented in the previous chapter and with Kuehne et al. (2017) and Huang et al. (2016), we extract Fisher vectors of improved dense trajectories over a temporal window of length 20 for each frame and reduce the result to 64 dimensions using PCA as described in Section 3.4.4. The visual model is a recurrent network with 64 gated recurrent units. In all experiments, the network is trained for $10,000$ iterations with a learning rate of 0.01 for the first $2,500$ iterations and 0.001 afterwards. We use stochastic gradient descent for optimization and set the minibatch size for backpropagation of the frames of a training sequence (cf. Section 7.2.2) to 512.

7.3.2 Robustness

We start with an evaluation of our proposed end-to-end learning algorithm. As discussed in Section 7.2.2, we enhance the loss function from Equation (7.5) by sampling additional frames from a buffer. In the following, we evaluate the impact of this enhancement and its parameters, namely number of sampled frames and buffer size.

Impact of Old Data Sampling. The first proposition to enhance the robustness of our algorithm is to maintain some recently seen sequences and their inferred labeling in a buffer and to sample a certain amount K of additional frames from this buffer. This way, we want to ensure that in each iteration, the overall data and class distributions are sufficiently

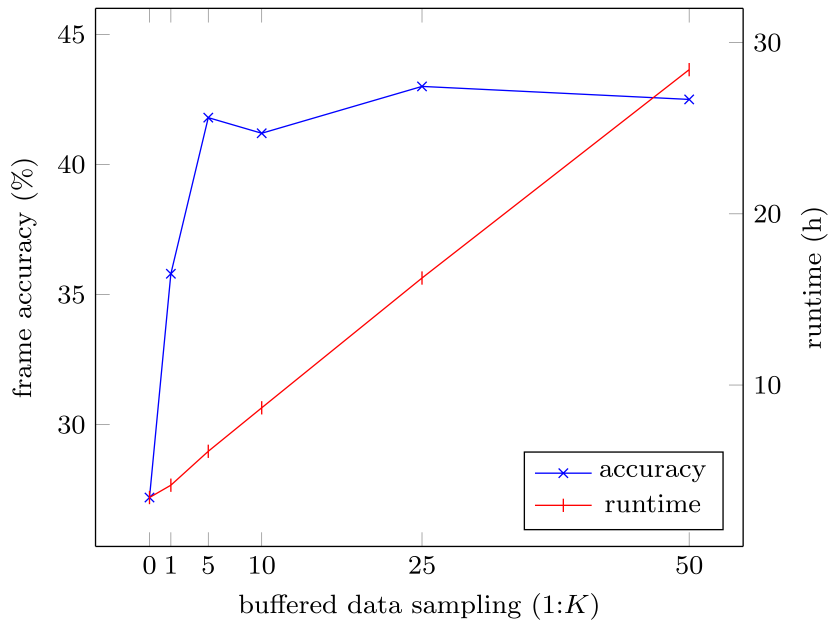
Algorithm 7.2 Reconstruction of the best segmentation

```
1: $(\ell, c) = \arg\max_{\ell,c} p(\ell|c) \cdot Q(T, \ell, c, h_{\text{end}})$
2: $h = h_{\text{end}}$
3: classes $= [\,]$
4: lengths $= [\,]$
5: $\ell_{\text{old}} = 0$
6: for $t = T, \dots, 1$ do
7:   if $\ell_{\text{old}} \leq \ell$ then
8:     classes.append($c$)
9:     lengths.append($0$)
10:  lengths.back() $=$ lengths.back() $+ 1$
11:  $\ell_{\text{old}} = \ell$
12:  $(\ell, c, h) = A(t, \ell, c, h)$
13: return classes.revert(), lengths.revert()
```

well captured. For the purpose of analyzing which value for K is necessary, we assume an unlimited buffer size, i.e. all previously processed sequences are maintained in memory. The results are illustrated in Figure 7.5. If we do not sample from previously seen sequences, the model is learned online, i.e. the training sequences are directly processed and not stored in a buffer. In this case, our approach achieves a frame accuracy of 27.2%. If we use a buffer and sample frames from it, the accuracy is greatly increased. Without sampling from the buffer, the model learns a strong bias towards the characteristics and class distributions of the current video only. This can be avoided by adding other frames from different classes and sequences to the loss function. While a 1:1 sampling, i.e. for each frame in the sequence one buffered frame is sampled, already shows a huge improvement, we find the optimization to stabilize at a sampling rate of 1:25. Thus, we stick to this value in all remaining experiments.

Impact of the Buffer Size. For the above evaluation, we assumed an unlimited buffer size, i.e. every processed sequence could be stored. This may be undesirable in case of large datasets for two reasons: first, depending on the amount of training data, it can be prohibitive to maintain all videos in memory at the same time. Second, the underlying assumption when using the buffer is that the frame/label pairs that are sampled are still more or less consistent with the current model. While this assumption is reasonable if all buffered sequences have been processed only a few iterations ago, it will certainly be wrong if there are frame/label pairs that have been generated by a model a few thousand iterations ago. Hence, we evaluate the impact of the buffer size on the performance, see Figure 7.6. Since we already fixed a sampling ratio of 1:25, a buffer size of less than 25 sequences is not reasonable. A too small buffer of less than 100 sequences does not reflect the overall data and class distributions well enough, resulting in a poor segmentation performance. With more than 200 buffered sequences, however, the system stabilizes. Considering the size of the datasets we use (less than $2,000$ sequences each), old frame/label pairs being inconsistent with the current model are not an issue here. Hence, we leave the buffer size unlimited for the remainder of this chapter.

Batch Size. In all experiments, we use a batch size of one. Figure 7.7 shows that with

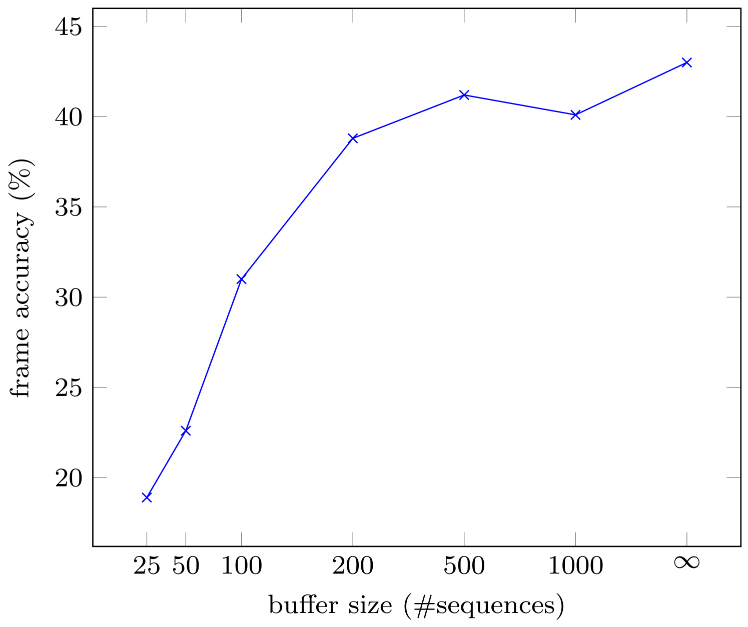


ratio 1:K	–	1:1	1:5	1:10	1:25	1:50
accuracy	27.2	35.8	41.8	41.2	43.0	42.5
runtime (h)	3:26	4:09	6:08	8:40	16:15	28:25

Figure 7.5: Impact of buffered data sampling. A sampling ratio of 1:K means that for each frame of the current sequence, K buffered frames are sampled. The first column shows the result for online learning, i.e., without a buffer. Runtime is measured on a K80.

larger batch sizes the accuracy slowly drops. Our model is continuously updated, i.e. segmentation information from previous iterations enters the parameter updates, via a running length and prior estimate as well as through buffered data. Thus, a small batch size allows for a rapid adaptation of the length model and prior.

Convergence Behavior. Figure 7.8 shows the convergence behavior of our algorithm as a pure online learning approach (no buffered data sampling) and with the robustness enhancements, i.e. with a 1:25 data sampling and an unlimited buffer size. While both variants of our algorithm start to converge after 2,000 to 3,000 iterations, the robustness enhancement is particularly advantageous at the beginning of training, adding a huge margin in terms of frame accuracy compared to the pure online variant. Note that the approach presented in the previous chapter suffers from oscillating accuracies over the iterations of the two-step scheme, cf. Figure 6.14. The proposed NN-Viterbi, in contrast, has a smooth and stable convergence behavior for both variants.



buffer size	25	50	100	200	500	1000	∞
accuracy	18.9	22.6	31.0	38.8	41.2	40.1	43.0

Figure 7.6: Impact of the buffer size for a buffered data sampling ratio of 1:25. Only a few hundred buffered sequences are already sufficient for robust learning.

7.3.3 Impact of Direct Learning and Model

In this section, we evaluate the impact of our proposed algorithm compared to the state-of-the-art methods for weakly supervised learning which generate pseudo ground truth instead of using the transcript annotations directly for learning as discussed in Section 7.2.1, and the advantages of temporal modeling using an explicit length model rather than an HMM as discussed in Section 7.2.3. The results are shown in Table 7.1.

7.3.3.1 Temporal Modeling: HMM vs. Length Model

Since the approach presented in Chapter 6 and many other recent methods use a hidden Markov model for the temporal progression throughout the sequence (Kuehne et al., 2017; Koller et al., 2017), we first show the benefits of modeling the temporal progression directly with a length model. Although the Viterbi decoding is more involved in this case, it allows to train a model directly on the action classes rather than on hidden Markov model states. First, note the impact of temporal modeling in general: if we neither use an HMM nor an explicit length model, the accuracy drastically drops, see first row of Table 7.1. When introducing an HMM as in Chapter 6, nearly +10% improvement can be observed. Using the

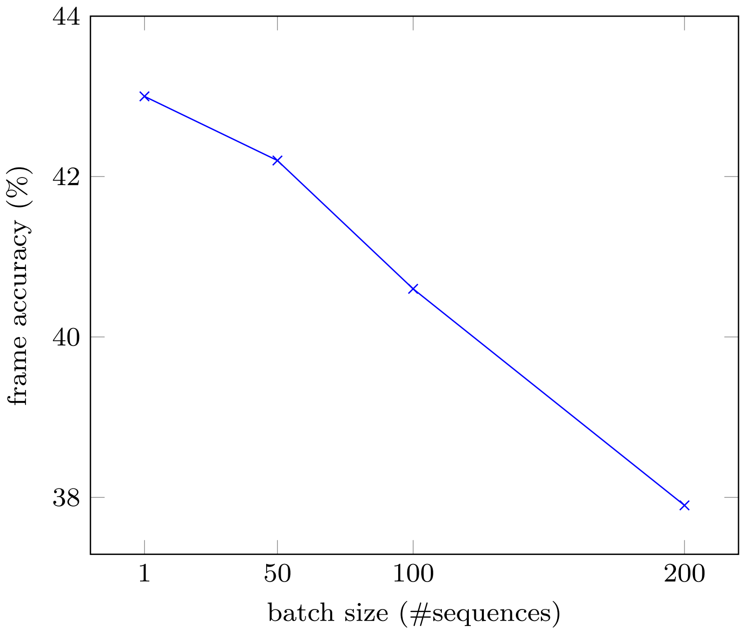


Figure 7.7: Effect of the batch size. A small batch and frequent updates are beneficial for better accuracy.

factorization from Equation (7.12) with the explicit length model, however, a further gain of +6% is achieved, see fourth row of Table 7.1. The reason for the latter is twofold. First, the training data is aligned to the actual classes rather than to a huge number of HMM states, so for each class more training examples are available. Second, the number of HMM states is fix during network training, while the length model can dynamically adapt to the learned model during training. Notably, using a length model on HMM states is not recommendable since HMM states are typically of very short duration and the state-wise length model has no major impact.

	accuracy (%)	runtime (h)
pseudo ground truth (Chapter 6)	23.9	03:45
pseudo gr.-tr. + HMM (Chapter 6)	33.3	08:12
pseudo gr.-tr. + HMM + length regularization (Chapter 6)	36.4	17:21
pseudo gr.-tr. + length model	39.1	06:04
NN-Viterbi + length model	43.0	22:43

Table 7.1: Impact of length modeling in combination with NN-Viterbi compared to different models using a pseudo ground truth. Training time is measured on a TitanX.

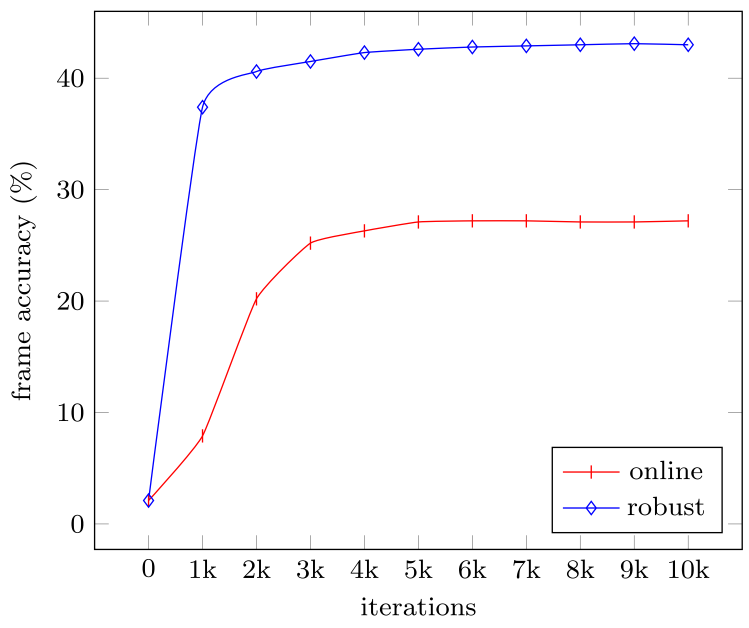


Figure 7.8: Convergence behavior of our NN-Viterbi algorithm in both variants, online (red) and with enhanced robustness (blue), over 10,000 training iterations.

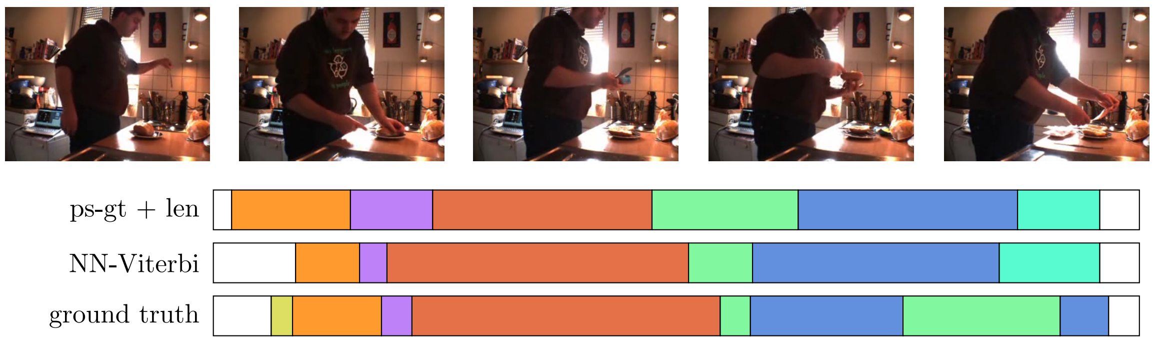
7.3.3.2 Pseudo Ground-Truth vs. Direct Learning

Note that so far, the model is still trained according to the two-step paradigm of repeatedly generating a pseudo ground truth and optimizing the network. Using the proposed NeuralNetwork-Viterbi algorithm, on the contrary, leads to much better results of 43.0% accuracy, which can be attributed to the direct loss, see Table 7.1. In case of the two-step scheme, the model is encouraged to learn the errors that are present in the generated pseudo ground truth. This can be avoided by including the transcripts directly into the model learning.

In Figure 7.9, two example segmentations are shown. Recall that for the two-step scheme, the initial pseudo ground truth is a uniform segmentation. Even after several iterations, a bias towards uniform sequence lengths can be observed. This leads to inaccurate detections of short segments (upper example segmentation) or even completely missed segments (lower example segmentation). The proposed NN-Viterbi learning is much more accurate, specifically when the segment lengths vary strongly.

7.3.4 Incremental Learning

In a classical learning setup, usually a fixed training set is provided. In this case, it is convenient to process all data in random order. For algorithms working in an online or incremental fashion, however, an interesting practical question is what happens if not all training data is available right at the beginning. For instance, video data from different actors is added to the training data over time. Or, training data for some classes is only



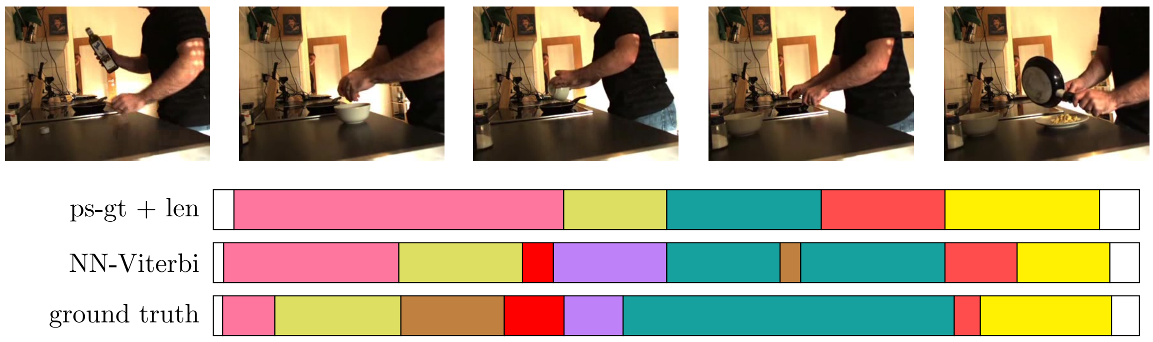


Figure 7.9: Example segmentations of two videos from the Breakfast dataset. The two-step scheme with pseudo ground truth and length model has a bias towards uniform lengths, which prevents short actions from being detected accurately. The NN-Viterbi approach is much more robust.

available at a later point in time.

We therefore analyze our algorithm under such conditions. To this end, we sorted the training set (a) by the ten coarse Breakfast activities[1] and (b) by the actors, see Table 7.2. In the first case, coarse activities that have been observed in the beginning, e.g. *cereals* and *coffee*, hardly lose any accuracy compared to training with randomly shuffled data, see Figure 7.10. Later coarse activities are usually not learned well and experience a relative drop of about 50% compared to random shuffling. The comparably small performance drop for *milk* and *tea* is due to the fact that these activities share a lot of fine-grained action classes with *cereals* and *coffee*, for instance `take_cup` or `pour_milk`.

Compared to the case where all data is available right at the beginning and random shuffling is possible, sorting the data by actor still results in a very good performance. Apparently, learning the correct class distributions right at the beginning is very important, while changes in appearance over time - such as changing actors - still allows to robustly

[1]Recall that each video in the Breakfast dataset belongs to one of ten coarse activities. The activities are compositions of 48 fine-grained action classes.

		frame accuracy (%)
(a)	sorted by activity	27.9
(b)	sorted by actor	41.5
(c)	randomly shuffled	43.0

Table 7.2: Impact of the sequence input order on the robustness of the algorithm. The videos are sorted (a) by the ten coarse activities of the Breakfast dataset, (b) by the performing actor, and (c) randomly.

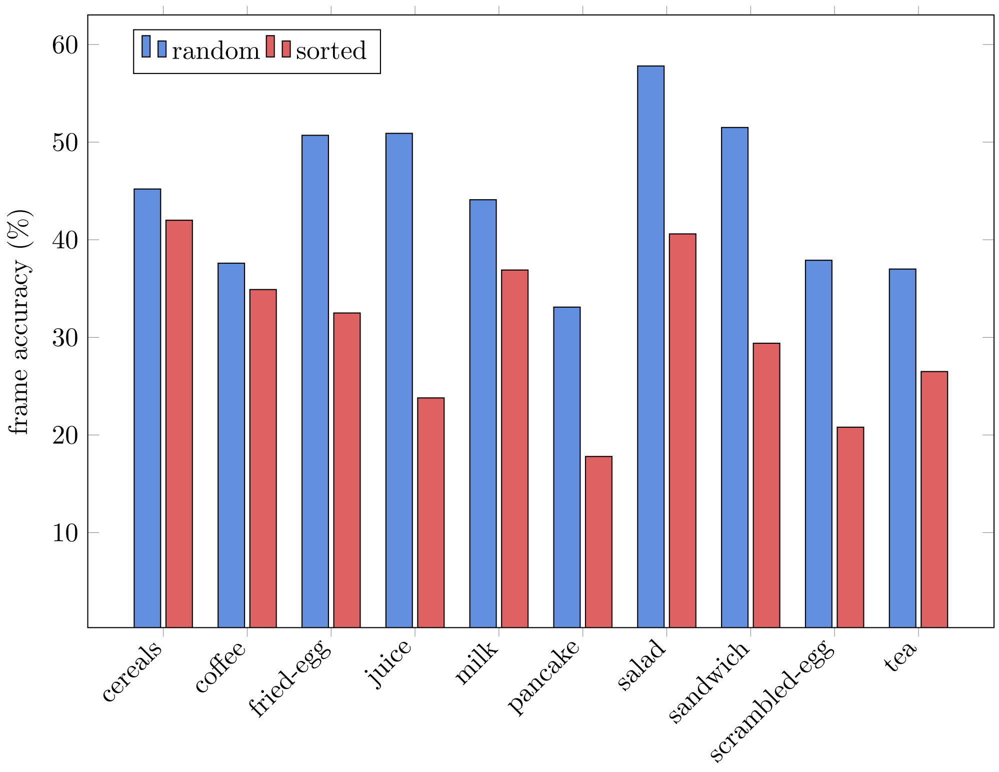


Figure 7.10: Accuracy per coarse activity for randomly shuffled training data and training data sorted by coarse activities. Left activities have been seen early during training, right activities later.

	frame accuracy (%)	
	Breakfast	**50 Salads**
OCDC (Bojanowski et al., 2014)	8.9	–
CTC (Huang et al., 2016)	21.8	11.9
HTK (Kuehne et al., 2017)	25.9	24.7
ECTC (Huang et al., 2016)	27.7	–
pseudo gr.-tr + HMM (Chapter 6)	33.3	45.5
pseudo gr.-tr. + HMM + length regularization (Chapter 6)	36.4	–
TCFPN (Ding and Xu, 2018)	38.4	–
NN-Viterbi	**43.0**	**49.4**

Table 7.3: Comparison of our method to several state-of-the-art methods for the task of temporal action segmentation. Results are reported as frame accuracy (%).

learn the underlying concepts of the classes.

7.3.5 Comparison to State-of-the-Art

In this section, we compare our approach to state-of-the-art methods for the same task, see Table 7.3. While OCDC (Bojanowski et al., 2014) is based on a discriminative clustering, the approach of Kuehne et al. (2017) and the method proposed in Chapter 6 rely on hidden Markov models and train their systems with the classical repeated two-step scheme. The model formulation is similar to the factorization from Equation (7.12). Still, NN-Viterbi outperforms the methods by a large margin. CTC and ECTC allow to optimize the posterior probabilities $p(\mathbf{c}_1^N|\mathbf{x}_1^T)$ directly. However, the criterion does not include explicit models such as a stochastic grammar or a length model. The assumption is that the underlying recurrent network can learn all temporal dependencies on its own. As also shown in Huang et al. (2016), this can lead to degenerate segmentations particularly when videos are long, since even LSTMs usually struggle to memorize context over multiple hundred frames. Particularly if the temporal dynamics change quickly, LSTM based approaches suffer. A recent theoretical analysis shows that only temporal data with limited dynamics can be modeled effectively using LSTMs (Tallec and Ollivier, 2018). Human actions typically are rather long, hence modeling context and length explicitly is very important and purely CTC based methods struggle to achieve comparable performance. Lin et al. (2017a) also use a CTC based model on Breakfast to infer the sequence of actions in a video. They evaluate the unit accuracy, i.e. the edit distance between the inferred action transcript and the ground truth transcript, and obtain 43.4% unit accuracy. With our approach, we obtain 55.5% unit accuracy.

7.3.6 Action Alignment

In contrast to the previous task, the ordered action sequences $\mathbf{c}_1^N$ are now also given for inference. Thus, only the alignment of actions to frames, or in other words, the lengths $\mathbf{l}_1^N$

Hollywood Extended	
	frame accuracy (%)
OCDC (Bojanowski et al., 2014)	43.9
HTK (Kuehne et al., 2017)	46.0
ECTC (Huang et al., 2016)	41.0
pseudo gr.-tr. + HMM (Chapter 6)	46.3
pseudo gr.-tr. + HMM + length regularization (Chapter 6)	46.0
TCFPN (Ding and Xu, 2018)	39.6
NN-Viterbi	**48.7**

Table 7.4: Comparison of our method to several state-of-the-art methods for the task of action alignment. Results are reported as a variant of the Jaccard Index (intersection over detection).

of each segment, need to be inferred. The training procedure is exactly the same as before.

The results are shown in Table 7.4. Neither the method of Ding and Xu (2018) nor the ECTC approach of Huang et al. (2016) yield competitive results on this task, albeit being very strong on the segmentation task. Both of our approaches, the fine-to-coarse model from Chapter 6 as well as the here proposed NeuralNetwork-Viterbi, show good performance for action alignment. Overall, NeuralNetwork-Viterbi outperforms the second best method on this task – the pseudo ground truth based RNN/HMM from the previous chapter – by 2.4%.

7.4 Semi-supervised Learning

We discussed fully supervised learning in Chapter 5 and proposed two systems for weakly supervised learning in Chapter 6 and in this chapter. The advantages and disadvantages of each approach are easily summarized: while full supervision allows for stronger models, the annotation is expensive to obtain. Weak annotations are easy to obtain but the quality of the models is not as good.

A compromise between the two extremes is to provide more annotation than just action transcripts but less than a full frame-level annotation. Therefore, we use sparse frame annotations of the training videos. Such annotation is stronger than pure action transcripts but still easy to obtain. Movie scripts and subtitles, for instance, can be used to automatically obtain temporal annotations in videos (Duchenne et al., 2009). While it is difficult to obtain precise temporal frame-level labels from such data, sparse frame annotations in the video are easier to get. Even for human annotators, labeling a single frame in a video by looking at a short snippet around this frame as proposed in Gu et al. (2018) is easier and faster than annotating precise temporal action boundaries.

We refer to the training of systems utilizing sparse frame-level annotations as semi-supervised learning. Note that we still keep the action transcripts as supervision, such that the semi-supervised setting provides strictly more supervision than action transcripts alone.

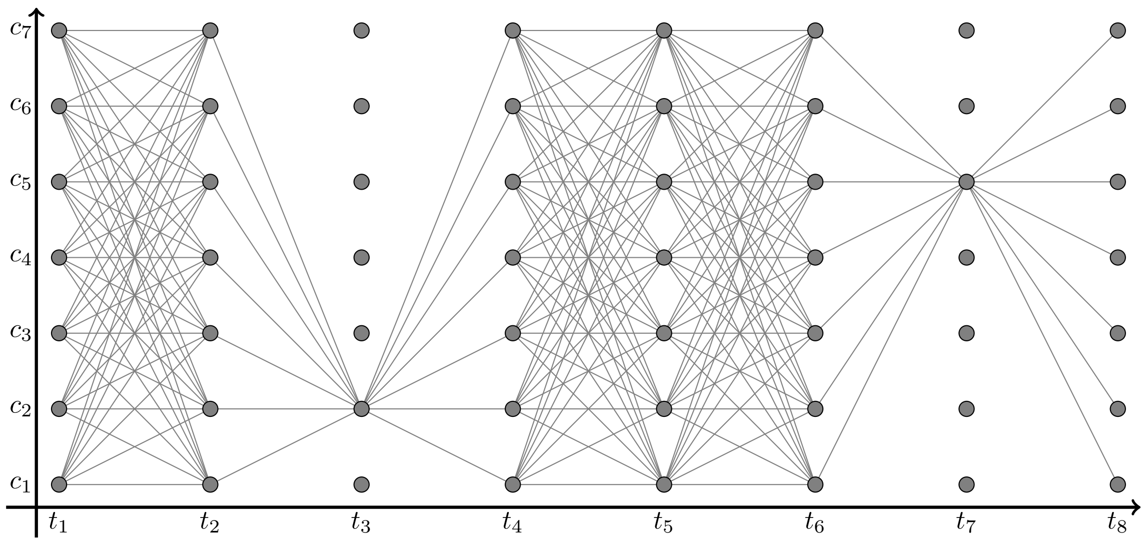
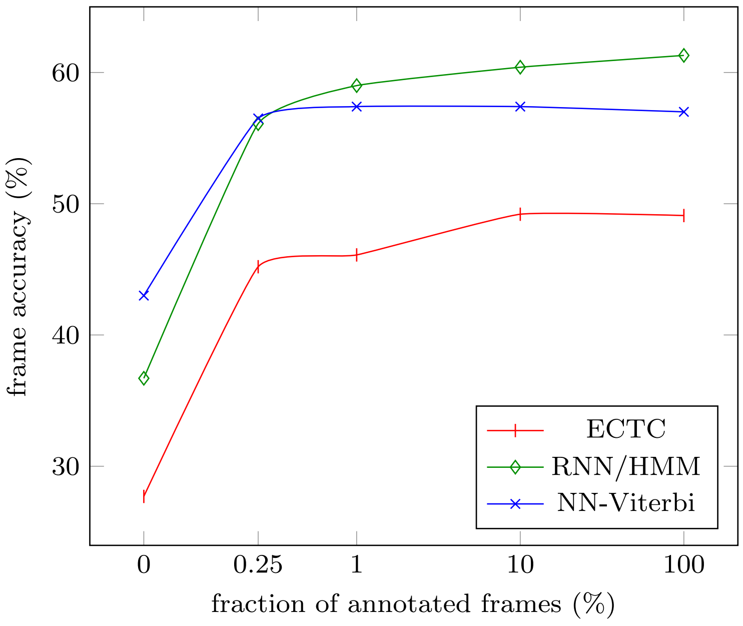


Figure 7.11: In the semi-supervised setting, the Viterbi path computed during training is forced to go through the annotated frame/label pairs, (t_3, c_2) and (t_7, c_5) in this example. This can be achieved by setting the probabilities of the visual model to zero for all but the annotated classes.

If the action transcripts were excluded from training, the model could hypothesize an arbitrary amount of non-existing action instances between two frame annotations. In practice, semi-supervised learning can be easily implemented in the NeuralNetwork-Viterbi algorithm by forcing the Viterbi path to go through the annotated frame/label pairs at training time, i.e. for an annotation pair (t, c), the path has to go through class c at time t, see Figure 7.11 for an illustration. For the RNN/HMM approach from Chapter 6, given an annotation (t, c), the Viterbi path is allowed to go through all HMM states associated with class c, i.e. $s_1^{(c)}, \dots, s_{K_c}^{(c)}$ at time t. All other HMM states must not be used.

We compare the NeuralNetwork-Viterbi approach and the RNN/HMM with ECTC (Huang et al., 2016), who used the same semi-supervised setting on the Breakfast dataset. The original weakly supervised systems use no sparse frame annotation at all. The results in Figure 7.12 show the effect of gradually increasing the amount of annotated frames. A ratio of 0.25% annotated frames per video corresponds to a single annotated frame per action instance on average and a ratio of 100% is a fully supervised setting. The frame annotations are uniformly distributed over each training video. It can be seen that even with an average of one frame per action (0.25% annotation ratio), frame accuracy significantly increases. Moreover, both our approaches outperform the fully supervised ECTC algorithm already with only 0.25% annotated frames.

Also note that NeuralNetwork-Viterbi performs much better than the RNN/HMM approach in the purely weakly supervised setting but does not benefit as much from additional supervision. One reason is that the explicit length model used for NeuralNetwork-Viterbi is more robust than an HMM given the uncertainty in the weakly supervised setting. With an



	annotated frames (%)				
	–	0.25%	1%	10%	100%
	frame accuracy (%)				
ECTC (Huang et al., 2016)	27.7	45.2	46.1	49.2	49.1
RNN/HMM (Chapter 6)	36.7	56.1	**59.0**	**60.4**	**61.3**
NN-Viterbi	**43.0**	**56.5**	57.4	57.4	57.0

Figure 7.12: Evaluation of semi-supervised learning on the Breakfast dataset. An increasing amount of annotated frames leads to better frame accuracies. Both of our systems, RNN/HMM and NeuralNetwork-Viterbi, outperform the ECTC approach of Huang et al. (2016) by a large margin.

increasing amount of supervision, however, the HMMs allow for a more subtle modeling of temporal progressions within an action.

When annotating only 1% of the frames, the RNN/HMM approach is the best among the three and the frame accuracy is only 2.3% lower than in the fully supervised case. Thus, sparse frame-level annotations prove to be effective to boost temporal segmentation performance without the need for exhaustive manual annotation of each frame.

7.5 Going Back to Full Supervision

Taking the semi-supervised approach to its extreme by annotating every single frame eventually results in a fully supervised setup. Among the analyzed semi-supervised approaches, the RNN/HMM introduced in Chapter 6 performs best when full frame-level annotation is

	frame accuracy (%)
HTK (Kuehne et al., 2016)	56.3
TCFPN (Ding and Xu, 2018)	52.0
Approach from Chapter 5	55.8
fully supervised RNN/HMM	61.3

(a) Comparison of fully supervised methods on Breakfast. Results are reported as frame accuracy.

	Overlap				
	0.1	0.2	0.3	0.4	0.5
Structured Segment Networks (Zhao et al., 2017)	66.0	59.4	51.9	41.0	29.8
Re-thinking Faster-RCNN (Chao et al., 2018)	59.8	57.1	53.2	48.5	42.8
Approach from Chapter 5	63.8	60.3	55.7	47.2	35.8
fully supervised RNN/HMM	49.1	42.7	36.3	30.0	23.3

(b) Comparison of fully supervised methods on Thumos. Results are reported as mAP for different overlap ratios.

Table 7.5: Comparison of the fully supervised approach from Chapter 5 and the fully supervised RNN/HMM on (a) the Breakfast dataset and (b) Thumos.

used. In Chapter 5, we proposed another fully supervised approach that was not based on an RNN/HMM visual model but used a segment classifier instead.

In this section, we compare both approaches on Thumos and the Breakfast dataset. The datasets have complementary properties, cf. Section 2.4. Thumos contains a large fraction for background frames and a highly varying amount of action instances per video. Further, in most cases the action instances in a video are all of the same class. Breakfast, on the contrary, has dense action annotations and background frames only at the beginning and at the end of the video. The action instances in a video are of different classes and there are long-range dependencies between them.

Table 7.5a shows the comparison of both methods on the Breakfast dataset. For comparison, the results of the best performing related work are also added to the table. Since Breakfast has similar properties as 50 Salads and MPII Cooking, we use the length model that performed best on these two datasets in the approach from Chapter 5, i.e. the class-dependent Poisson distribution. Note that a trigram does not generate good results on Breakfast since the long-range dependencies are not well captured. Recall the definition of the perplexity from Equation (5.34), which can be interpreted as the average number of label choices the context model has per position. A low value indicates a high likelihood of the data under the given trigram model. While the perplexity for a trigram is only 1.203 on Thumos, it is 5.175 on Breakfast. Therefore, we use the same grammar as in Chapter 6 instead of a trigram context model on the Breakfast dataset. The fully supervised version of the RNN/HMM clearly outperforms the approach from Chapter 5. This is mainly due to the segment classifier that is used as visual model in Chapter 5. Breakfast is strongly motion driven and requires a thorough temporal modeling, which is not provided by the segment classifier due to the temporal averaging over the segment features.

Table 7.5b shows the same comparison on Thumos. Again, we provide the results for the best performing related approaches as a comparison. Note that we use a trigram as context model, which is, given its low perplexity, a very strong context model on that dataset. The situation in this set of experiments is different than for Breakfast. The approach from Chapter 5 clearly outperforms the fully supervised RNN/HMM. This can be attributed to the characteristics of Thumos. The dataset is strongly appearance driven, so a fine-grained temporal modeling is not as important as on Breakfast. Moreover, the segment classifier as visual model is already extremely strong, see Table 5.7. The RNN/HMM, however, only reaches a frame accuracy of 21% on Thumos.

7.6 Summary

We have proposed a direct learning algorithm that can handle weakly labeled video sequences. The algorithm is generic and can be applied to any kind of model whose best segmentation can be inferred using a Viterbi-like algorithm. Unlike the CTC criterion, our approach allows to include multiple explicitly modeled terms such as a context model and a length model, what has been proven crucial for good performance. Moreover, we showed that using an explicit length model and optimizing the video classes directly leads to a huge improvement over related HMM-based methods that use a pseudo ground truth. The proposed approach is not bound to weak supervision but can also be trained in a semi-supervised manner with sparsely labeled frames. Overall, the method outperforms other state-of-the-art approaches by a large margin and shows a robust and stable convergence behavior.

Chapter 8

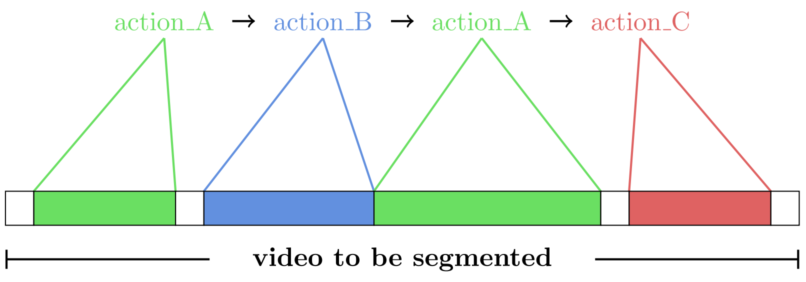
Action Sets

In Chapter 6 and Chapter 7, we introduced effective approaches for temporal action segmentation that only require weak supervision in form of action transcripts. With the goal of analyzing the vast amount of video data available on the Internet, however, there is need for approaches that can learn with even less supervision. In this chapter, we propose a method that learns from unordered action sets only instead of ordered action transcripts. While the performance is clearly worse than for fully supervised approaches or approaches that use action transcripts, we can still show remarkable accuracy considering the tremendous lack of temporal or frame-level annotation.

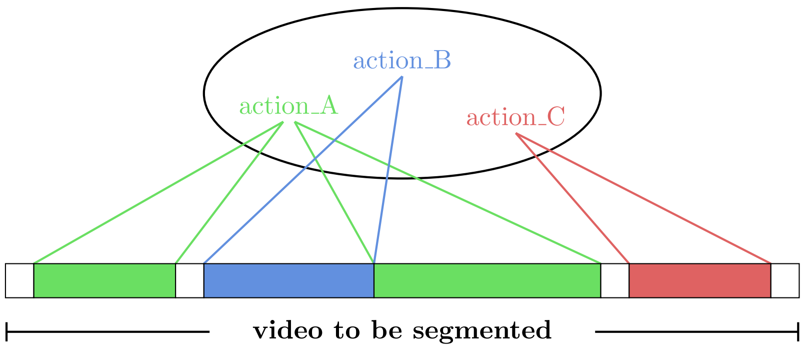
Contents

8.1 Introduction

The previously introduced methods that are focusing on weakly supervised learning based on action transcripts are a good step towards using annotations that are easy to obtain even for a large amount of video data. Still, when it comes to automatic generation of actions, such methods quickly reach their limits. Consider, two people discussing the weather while



(a) weak supervision: ordered action sequences



(b) weak supervision: action sets

Figure 8.1: (a) Weak supervision with ordered action sequences as proposed in Chapter 6 and Chapter 7 and also used by Bojanowski et al. (2014); Kuehne et al. (2017); Huang et al. (2016); Ding and Xu (2018). The number of actions and their ordering is known. (b) Weak supervision with action sets. Neither action orderings nor the number of occurrences per action are provided.

preparing a meal. In this case, the activities do not correspond to the acoustic signal contained in the video. Using movie scripts, subtitles, or the output of an automatic speech recognition system therefore is not a reliable source to mine action transcripts. The remaining option – obtaining the action transcripts from human annotators – is infeasible if a vast amount of video data is required for training. Thus, another source of annotation is needed in order to process large amounts of video data that is posted on social networks or other video streaming sites such as YouTube. One possible source of labeling is meta-tags. While the acoustic content in a video can be semantically different from the visual content, meta-tags usually are assigned based on what is happening in the video, i.e. which actions are performed.

As a first step to learn from such unstructured information as meta-tags, we propose a weakly supervised method that can learn to temporally segment actions in a video from unordered action labels, which we refer to as action sets. In contrast to methods that rely on action transcripts (cf. Figure 8.1a), we assume that neither ordering nor number of occurrences of actions is provided during training. Instead, only a set of actions occurring within the video is given (cf. Figure 8.1b). This task is much more difficult than the case where ordered action transcripts are given. Consider, for instance, a video with T frames and a transcript of C ordered actions. Then, there are $\frac{(C+T)!}{C!T!}$ possible labelings for the video. If the actions are not ordered, there are already C^T possible labelings. For a very short video of 100 frames and $C = 5$, this means that using unordered action sets as supervision already allows for about 10^{60} times more possible labelings than when provided ordered action transcripts.

In order to deal with such an enormously large search space, we propose three model components that aim at decomposing the search space on three different levels of granularity. The coarsest level is addressed by a context model that restricts the space of possible action sequences. On a finer level, a length model restricts the durations of actions to a reasonable duration. On the lowest, most fine-grained level, a frame model provides class probabilities for each video frame.

We describe the technical details of the approach in Section 8.2. In an extensive evaluation in Section 8.3, we investigate the impact of each component within the system. Moreover, temporal segmentation and action labeling quality is evaluated on unseen videos alone and on videos with action sets given at inference time as additional supervision.

8.2 Technical Details

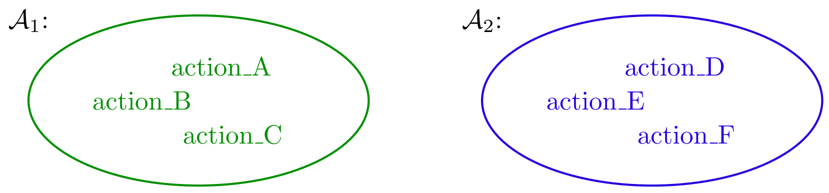
In the previous chapters, either full frame-level annotation or weak annotation in the form of action transcripts have been given for training. While action transcripts as supervision are widely used, see e.g. Chapter 6, Chapter 7, or the works of Bojanowski et al. (2014); Huang et al. (2016); Kuehne et al. (2017); Ding and Xu (2018), and doubtlessly easier to obtain than frame-level annotation, there are still scenarios where such kind of annotation is not available. Therefore, we reduce supervision further for such scenarios and rely on unordered sets of actions occurring in the video, see Figure 8.1b for an illustration. Neither the order of the actions nor the number of occurrences per action is known. Assuming the training set consists of I videos, then the supervision available for the i-th video is a set $\mathcal{A}_i \subseteq \mathcal{C}$ of actions occurring in the video. Such sets can in practice be mined from textual descriptions of the video or video tags.

During inference, no action sets are provided for the video and the model has to infer an action labeling from the video frames only. However, in many scenarios it is reasonable to assume that action sets can also be mined for unseen videos and thus, as an additional task, we also discuss the case where action sets are given for inference, see Section 8.3.7.

In order to address the task of action segmentation with action sets, we rely on the same general framework we used in the preceding chapters, i.e. a segmentation of a video is modeled as

$$(\hat{N}, \hat{\mathbf{c}}_1^{\hat{N}}, \hat{\mathbf{l}}_1^{\hat{N}}) = \operatorname*{arg\,max}_{N, \mathbf{c}_1^N, \mathbf{l}_1^N} \Big\{ p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N) \cdot p(\mathbf{l}_1^N | \mathbf{c}_1^N) \cdot p(\mathbf{c}_1^N) \Big\}, \tag{8.1}$$

where again $p(\mathbf{x}_1^T | \mathbf{c}_1^N, \mathbf{l}_1^N)$ is the visual model, $p(\mathbf{l}_1^N | \mathbf{c}_1^N)$ is a length model, and $p(\mathbf{c}_1^N)$ is the context model. We describe the concrete models for the three components in the following.



$$(\text{action\_B}, \text{action\_C}, \text{action\_B}, \text{action\_A}) \in \mathcal{L}(\Gamma_{\text{naive}})$$
$$(\text{action\_D}, \text{action\_F}) \in \mathcal{L}(\Gamma_{\text{naive}})$$
but
$$(\text{action\_A}, \text{action\_D}, \text{action\_F}) \notin \mathcal{L}(\Gamma_{\text{naive}})$$

Figure 8.2: Example of valid and invalid action sequences generated by the naive grammar Γ_{naive}. Only sequences constructed from elements of a single action set are valid.

8.2.1 Context Model

As mentioned before, the search space of possible segmentations is much larger when action sets are the only supervision available since the ordering constraints of action transcripts are not available. The first step to handle the huge search space is to restrict the possible action orderings using a grammar Γ in order to model the context prior $p(\mathbf{c}_1^N)$.

We rely on the right-regular grammar proposed in Section 6.3.2 and propose different strategies to create a useful grammar. In the following, let $\mathcal{L}(\Gamma)$ denote the language of Γ, i.e. the set of all action sequences that can be generated by Γ.

Naive Grammar

The most naive way of formulating a grammar that restricts the search space is to allow all sequences of actions that can be created using elements from each action set in the training data. Formally, this means

$$\mathcal{L}(\Gamma_{\text{naive}}) = \bigcup_{i=1}^{I} \mathcal{A}_i^*, \tag{8.2}$$

where i indicates the i-th training sample and $\mathcal{A}_i^*$ is the Kleene closure of $\mathcal{A}_i$. An action sequence $\mathbf{c}_1^N$ can be generated by Γ_{naive} if and only if there is an action set $\mathcal{A}_i$ in the training set such that all actions in $\mathbf{c}_1^N$ are in $\mathcal{A}_i$, see Figure 8.2.

Being a union of the Kleene closure of a finite set, $\mathcal{L}(\Gamma_{\text{naive}})$ is a regular expression and thus can be expressed by a right-regular grammar. Explicitly constructing the grammar is straightforward. The set of non-terminals is a set of I symbols h_i, one for each action set $\mathcal{A}_i$. The rules for each action set have to allow to produce each label $c \in \mathcal{A}_i$ as the first label, i.e. from the start symbol, as last label, i.e. without a new context symbol, or somewhere in between. Thus, for each class c from each action set $\mathcal{A}_i$, the rules

$$\begin{aligned} h_\$ &\overset{1.0}{\rightarrow} c\ h_i, \\ h_\$ &\overset{1.0}{\rightarrow} c, \\ h_i &\overset{1.0}{\rightarrow} c\ h_i, \\ h_i &\overset{1.0}{\rightarrow} c \end{aligned}$$

are added to the grammar.

Monte-Carlo Grammar

As a second choice, we provide a set of randomly sampled candidate action sequences. Therefore, we randomly generate a large amount of k action sequences. Each sequence is generated by randomly choosing a training sample $i \in \{1, \dots, I\}$. Then, actions are uniformly drawn from the corresponding action set $\mathcal{A}_i$ until the accumulated estimated means λ_c of all drawn actions exceed the video length T_i, i.e. new actions are drawn until for the sequence $\mathbf{c}_1^N$,

$$\sum_{n=1}^{N} \lambda_{c_n} > T_i, \tag{8.3}$$

where the mean lengths λ_c are estimated action class durations, see Section 8.2.2.

Once a set of randomly drawn action sequences is created, generation of the grammar follows the same steps as the generation of the grammar from given action transcripts that is illustrated in Figure 6.5.

Text-Based Grammar

Frequently, it is possible to obtain a grammar from external text sources, e.g. from web recipes or books. Given some natural language texts, we enhance the monte-carlo grammar by mining frequent word combinations related to the action classes. Consider two action classes v and w, for instance `butter_pan` and `crack_egg`. If either of the words `butter` or `pan` is preceding `crack` or `egg` in the textual source, we increase the count $N(v, w)$ by one. This way, word conditional probabilities

$$p(w|v) = \frac{N(v, w)}{\sum_{\tilde{w}} N(v, \tilde{w})} \tag{8.4}$$

are obtained that have a high value if v precedes w frequently and a low value otherwise. The actual construction of the grammar follows the same protocol as the monte-carlo grammar with the only difference that the actions are not drawn uniformly from the action set but according to the distribution $p(w|v)$, where v is the previously drawn action class.

8.2.2 Length Model

While a grammar already introduces some ordering constraints, the search space is still tremendously large, considering that actions can be of arbitrary and even practically unreasonable durations. Therefore, as a second step, we estimate a length model out of the scarce information we get from the training data. In order to model the length factor $p(\mathbf{l}_1^N|\mathbf{c}_1^N)$, we assume conditional independence of each segment length and further drop the dependence on all class labels but the one of the current segment, i.e.

$$p(\mathbf{l}_1^N|\mathbf{c}_1^N) = \prod_{n=1}^{N} p(\ell_n|c_n). \tag{8.5}$$

Each class-conditional $p(\ell|c)$ is modeled with a Poisson distribution for class c.

For the estimation of the class-wise Poisson distributions, only the action sets $\mathcal{A}_i$ provided in the training data can be used. Ideally, the free parameter of a Poisson distribution, λ_c, should be set to the mean length of action class c. Since this can not be estimated from the action sets, we propose two strategies to approximate the mean duration of each action class.

Naive Length Estimation

In the naive approach, the frames of each training video are assumed to be uniformly distributed among the actions in the respective action set. The average length per class can then be computed as

$$\lambda_c = \frac{1}{|\mathcal{I}_c|} \sum_{i \in \mathcal{I}_c} \frac{T_i}{|\mathcal{A}_i|}, \tag{8.6}$$

where $\mathcal{I}_c = \{i : c \in \mathcal{A}_i\}$ and T_i is the length of the i-th video. Note that in this formulation, it is assumed that each action from the action set occurs only once in the video. This is usually not true in practice, so the estimated mean length is easily overestimated.

Loss-based Length Estimation

The drawback of the naive approach is that actions that are usually short are assumed to be longer if the video is long. Instead, we propose to estimate the mean of all classes together. This can be accomplished by minimizing a quadratic loss function,

$$\sum_{i=1}^{I} \sum_{c \in \mathcal{A}_i} (\lambda_c - T_i)^2 \quad \text{subject to } \lambda_c > \ell_{\min}, \tag{8.7}$$

where $\ell_{\min}$ is a minimal action length. For minimization, we use constrained optimization by linear approximation (COBYLA Powell (1994)).

Again, λ_c can easily be overestimated since actions may occur multiple times in a video. However, this can not be included into the length model since the action sets do not provide such information.

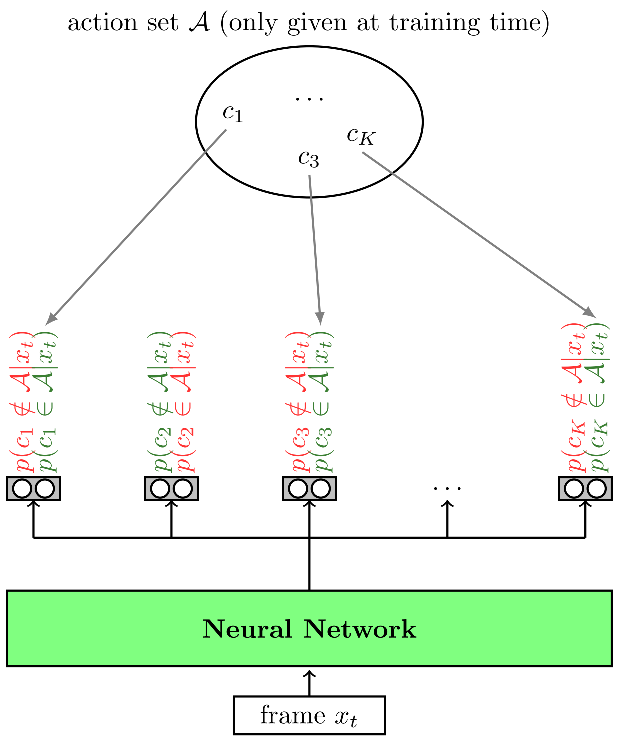

action set $\mathcal{A}$ (only given at training time)
c_1
$\dots$
c_K
c_3
$p(c_1 \notin \mathcal{A}|x_t)$
$p(c_1 \in \mathcal{A}|x_t)$
$p(c_2 \notin \mathcal{A}|x_t)$
$p(c_2 \in \mathcal{A}|x_t)$
$p(c_3 \notin \mathcal{A}|x_t)$
$p(c_3 \in \mathcal{A}|x_t)$
$p(c_K \notin \mathcal{A}|x_t)$
$p(c_K \in \mathcal{A}|x_t)$
$\dots$
Neural Network
frame x_t

Figure 8.3: The neural network used in the visual model. A single input frame is mapped to binary output layers for each class. During training, the network is optimized to predict the presence of all classes occurring in the complete video from the single input frame, i.e. given the action set $\mathcal{A}$ of the video, $p(c \in \mathcal{A}|x_t)$ is optimized to be high if $c \in \mathcal{A}$. On the contrary, if $c \notin \mathcal{A}$, the opposite outcome $p(c \notin \mathcal{A}|x_t)$ is optimized to be high.

8.2.3 Visual Model: Multi-task Learning of Action Frames

Given the grammar and the length model that already strongly restrict the search space, the last missing factor is the visual model providing a likelihood for each class to be present in a given frame.

In order to model this last remaining factor from Equation (8.1), we train a network with $|\mathcal{C}|$ many binary softmax outputs, see Figure 8.3. Each layer predicts if for a given frame x_t label c is among the actions occurring in the video or not. The idea behind it is similar to multiple instance learning. Action classes usually occur in different contexts. Particularly, consider two action sets $\mathcal{A}_1$ and $\mathcal{A}_2$ with $\mathcal{A}_1 \cap \mathcal{A}_2 = \{c\}$. When training the network, all frames from the first video associated with $\mathcal{A}_1$ learn a strong response on the classes from $\mathcal{A}_1$. A frame belonging to class c, however, does not contain information about the other classes, so those responses are mainly based on artifacts and video specific appearance. The same holds for classes from $\mathcal{A}_2$ and the associated video frames. For class c, however, the network needs to learn a high response for video frames of both videos, so there is an incentive to extract class specific features instead of video specific appearance. Overall, the network is more likely to learn class-specific features for the true class of a frame and non-generalizing, video specific responses for all other classes.

As loss of our network, we use the accumulated cross-entropy loss of each binary classification task. We define a class-posterior distribution based on the scores of the positive units of each binary output by renormalizing them,

$$p(c|x_t) := \frac{p(c \in \mathcal{A}|x_t)}{\sum_{\tilde{c}} p(\tilde{c} \in \mathcal{A}|x_t)}, \tag{8.8}$$

and transform them into class-conditional probabilities

$$p(x_t|c) \propto \frac{p(c|x_t)}{p(c)}. \tag{8.9}$$

Note that this requires an estimate of the class prior, which is not given by the annotation. We therefore use a heuristic. Let

$$\text{count}(c) = \sum_{i \in \{1,\dots,I: c \in \mathcal{A}_i\}} T_i \tag{8.10}$$

be the accumulated number of frames of videos containing class c. The heuristic class prior is then

$$p(c) = \frac{\text{count}(c)}{\sum_{\tilde{c}} \text{count}(\tilde{c})}. \tag{8.11}$$

The visual model is then obtained using Equation (8.9) assuming framewise independence,

$$p(\mathbf{x}_1^T|\mathbf{c}_1^N, \mathbf{l}_1^N) = \prod_{t=1}^{T} p(x_t|c_{n(t)}). \tag{8.12}$$

8.2.4 Inference

With the explicit models for each factor, the optimization problem from Equation (8.1) reduces to

$$(\hat{N}, \mathbf{c}_1^{\hat{N}}, \mathbf{l}_1^{\hat{N}}) = \underset{\substack{N, \mathbf{c}_1^N, \mathbf{l}_1^N, \\ \exists \mathbf{h}_1^N:\ h_{n-1} \rightarrow c_n\ h_n \in \mathcal{R}}}{\arg\max} \left\{ \prod_{t=1}^{T} p(x_t|c_{n(t)}) \cdot \prod_{n=1}^{N} p(\ell_n|c_n) \cdot p(c_n|h_{n-1}) \right\}, \tag{8.13}$$

which is the same optimization problem as in Section 7.2.4. Therefore, the best segmentation can be found with the same modification of the Viterbi decoding presented in the previous chapter.

8.3 Experiments

In this section, we analyze the components of our approach, starting with the grammar in Section 8.3.2 and the length model in Section 8.3.3, before we compare our system to existing methods that use more supervision in Section 8.3.6.

8.3.1 Setup

We evaluate our approach on the Breakfast dataset, MPII Cooking 2, and Hollywood Extended, see Chapter 2 for a detailed description. Breakfast is the most widely used dataset for weakly supervised action segmentation and allows to compare the here presented method to approaches that use more supervision. Hollywood Extended has different characteristics, particularly a much higher background fraction. Moreover, action classes frequently occur two or three times in a video. MPII Cooking 2, on the contrary, serves as an example for the limitations of action sets. It contains only a small number of videos that are all comparably long and contain a huge amount of action instances each. Following the common evaluation protocols, we report frame accuracy on Breakfast, the IoU-based Jaccard Index on Hollywood Extended, and mAP based on the midpoint hit criterion on MPII Cooking 2.

As video features, we use the same features as in the previous two chapters, i.e. 64-dimensional framewise Fisher vectors of improved dense trajectories as described in Section 3.4.4. If not mentioned otherwise, we use the monte-carlo grammar and the loss-based length model. The in-depth evaluation of our approach is conducted on Breakfast, results on other datasets are reported in Section 8.3.6.

While the motivation for action sets is the availability of meta-tags in web videos, for our evaluation we simulate the existence of action sets by taking the set of all actions that occur within a video as supervision. Mining action sets from meta-tags introduces noise and labeling errors, whereas taking the true action sets ensures that all actions occurring in the video are indeed present in the set and no other actions are mistakenly put into it. This allows for a clean and unbiased evaluation of the method and a fair comparison to methods that use stronger supervision as the same action classes are available.

For the neural network in the visual model we use a simple feed forward network with a single hidden layer of 256 rectified linear units. We train the model for two epochs only, finding that it converges quickly. Experiments with deeper models could not generalize to the test data (VGG-16: frame accuracy of 3.1% on Breakfast). We also evaluated the neural network based multiple instance learning approach of Wu et al. (2015a), which also was not able to make reliable predictions (accuracy 8.9% on Breakfast). We therefore found the multi-task network as proposed in Section 8.2.3 to be a simple yet effective model.

During inference, we allow to hypothesize new segments only every 30 frames. This allows for inference roughly in realtime without affecting the performance of the system compared to a more fine-grained segment hypothesis generation. An evaluation of different frame sampling rates is provided in Section 8.3.5.

8.3.2 Effect of the Grammar

The main contribution of the grammar is to limit the search space and remove unrealistic action sequences. We compare different kinds of grammars and report the frame accuracy on both, test and train set. Recall that due to weak supervision, our method does not necessarily provide good results on the training videos, making it interesting to investigate both sets. As shown in Table 8.1, the use of a sophisticated grammar is crucial for good performance.

	frame accuracy	
Grammar	train	test
none	14.7	9.9
naive	19.4	13.4
monte-carlo	28.2	23.3
manually created	33.3	26.9
ground truth	36.7	29.4

Table 8.1: Evaluation of the proposed method on Breakfast using different context-free grammars. As length model, the loss-based approach is used.

The naive grammar is only slightly better than the system without any grammar. The monte-carlo grammar boosts the frame accuracy by 10% on the test set. Note that we found the number of k monte-carlo samples for the grammar not to be critical and chose $1,000$ randomly generated sequences for all experiments. Using a ground truth grammar, i.e. a grammar learned from ordered action transcripts (which are not provided in our setting) gives an upper bound on the performance that can be reached by improving the grammar only. Notably, the monte-carlo grammar is only 6% below this upper bound.

As a further comparison, we asked a human annotator to manually create the grammar. We gave all action sets contained in the training data to an annotator who did not see the dataset before. The action sets have been provided in a text file with one set per line, the classes in each line being in alphabetical order. The annotator was instructed to generate a reasonable ordered action sequence out of these action sets. He was allowed to use each action multiple times and the only restriction we imposed was that all actions from the set need to occur at least once in the generated sequence. On Breakfast, with action sets of one to eleven elements, the annotator needed an average of one minute per set to generate an ordered action sequence.

From the list of reasonable action sequences provided by the annotator, we then generated a grammar as described in Section 6.3.2 and illustrated in Figure 6.5. This manually created grammar serves as a comparison of the purely data driven monte-carlo grammar to human knowledge. Although the manual grammar is better, the frame accuracy only differs by 3.6%. Since the annotator on average only needed one minute per action set, a manual grammar is also a cheap opportunity to add human knowledge without the need to actually annotate videos.

Grammars from Written Texts

As proposed in Section 8.2.1, textual sources can be used to enhance the monte-carlo grammar by restricting the transition between action classes to only the likely ones. We evaluate such a text-based grammar for all three datasets.

For the Breakfast dataset, each video is annotated with one out of ten coarse breakfast

	Breakfast	Cooking 2	Holl. Ext.
	frame acc.	*midpoint hit*	*jacc. idx*
monte-carlo	23.3	9.8	9.3
text-based	23.2	10.6	9.2

Table 8.2: Evaluation of the text-based grammar. For Cooking 2, where the text sources are closely related to the content of the videos, an improvement can be observed.

	frame accuracy	
Length model	train	test
naive	25.4	20.1
loss-based	28.2	23.3
ground truth	34.1	25.7

Table 8.3: Evaluation of the proposed method on Breakfast using different length models. As grammar, the monte-carlo approach is used.

activities such as *bowl of cereals* or *fruit salad*. We took those ten coarse activity classes as search terms on two recipe web pages, www.allrecipes.com and www.foodnetwork.com. For each search term and web page, we stored the first ten pages of retrieved recipes. In total, textual sources with 120,000 words have been obtained.

For Hollywood Extended, we downloaded ten movie scripts from the Internet Movie Script Database www.imsdb.com. The ten movies have been selected by going through IMDBs top-ranked movie list. If a script was not available for a movie, we skipped it and tried the next movie. In the end, all ten downloaded scripts are taken from IMDB top-twenty movies. The overall number of words in the scripts is 76,000.

On MPII Cooking 2, the authors already provide a textual description linked to the videos. More precisely, they used Amazon Mechanical Turk and asked people to give a tutorial-like description of certain kitchen tasks. Each description was limited to at most 15 steps. In total, the scripts contain 108,000 words. More details on the creation of the scripts can be found in the Cooking 2 dataset paper (Rohrbach et al., 2016).

The text sources used for Breakfast and Hollywood Extended are only loosely connected to the datasets, whereas the textual source for Cooking 2 covers exactly the same domain as the videos. Not surprisingly, we find that only for this case, the text-based grammar leads to an improvement over the monte-carlo grammar, see Table 8.2. For the other datasets, neither an improvement nor a degradation is observed.

8.3.3 Effect of the Length Model

Besides the choice of the context-free grammar, the length model is a crucial component of the system. The estimated mean action lengths influence the performance in two ways. First,

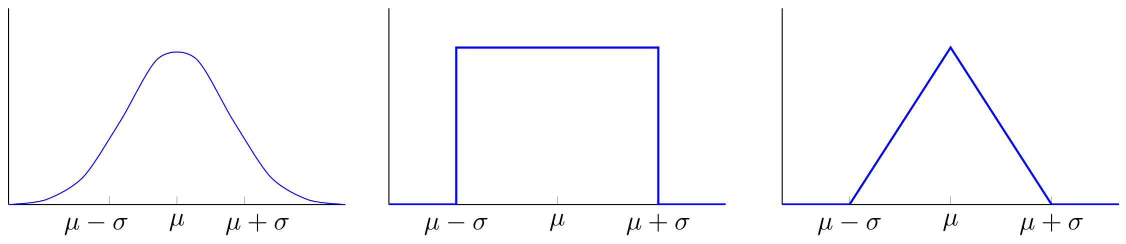


Figure 8.4: Three alternatives to the Poisson length model. Left: a Gaussian model with mean μ and standard deviation σ. Middle: a box function of width 2σ around the mean length μ. Right: A triangular model of width 2σ around the mean length μ.

they define the Poisson distribution that contributes to the actual length of hypothesized action segments. Second, they have a huge impact on the number of action instances that are generated for each action sequence in the monte-carlo grammar.

Mean Length Approximation

We compare the two proposed mean approximation strategies, naive and loss-based mean approximation, with a ground truth model, i.e. the true action means estimated on a frame-level ground truth annotation of the training data. The results are shown in Table 8.3. The naive mean approximation suffers from some conceptual drawbacks. Due to the uniform distribution of video frames among all actions occurring in the video, short actions may be assigned a reasonable length as long as the video is also short. If the video is long, however, short actions get the same share of frames as long actions, resulting in an over-estimation of the mean for short actions and an under-estimation of the mean for long actions. The loss-based mean approximation, on the contrary, can provide more realistic estimates by minimizing Equation (8.7). Note that the solution of the problem in principle would allow for negative action means. Hence, setting the minimal action length $\ell_{\text{min}} > 0$ is crucial. In practice, we want to ensure a reasonable minimum length and set $\ell_{\text{min}} = 50$ frames, corresponding to roughly two seconds of video. The loss-based mean approximation performs significantly better than the naive approximation, increasing the frame accuracy by 3%.

Comparing these numbers to the ground truth length model reveals that particularly on the train set, on which the ground truth lengths have been estimated, there is still room for improvement. Considering the small amount of supervision that we can utilize to estimate mean lengths, i.e. action sets only, and the small gap between the loss-based approach and the ground truth model on the test set, on the other hand, we find that our loss-based method already yields a good approximation.

Evaluating Different Length Models

So far we modeled the length with a Poisson distribution. There is a variety of other possible length models. In Figure 8.4, three additional models are illustrated, a Gaussian model, a box model, and a triangular model. The probability outside $[\mu-\sigma, \mu+\sigma]$ is zero for the box model

	Gaussian	Box	Triangle	Poisson
frame accuracy	0.148	0.220	0.227	0.233

Table 8.4: Evaluation of four different length models on the Breakfast dataset.

		frame accuracy	
grammar	length model	train	test
✗	✗	0.118	0.080
✗	✓	0.147	0.099
✓	✗	0.208	0.154
✓	✓	0.282	0.233
fully supervised		0.774	0.556

Table 8.5: The first four rows are a comparison of the impact of the grammar and the length model on the Breakfast dataset, the last is the proposed system trained on fully supervised, i.e. framewise annotated, data. It is an upper bound for the weakly supervised setup.

and the triangular model whereas the Gaussian model has infinite support. The standard deviation σ of each action class is heuristically estimated by mapping actions according to their mean length onto the possible segmentations generated by the monte-carlo grammar. For instance, if μ_c is twice as large as $\mu_{\tilde{c}}$ and there is a monte-carlo sequence containing both c and $\tilde{c}$, the c segment will be twice as large as the $\tilde{c}$ segment. Applying this mapping to all sampled monte-carlo sequences allows to estimate a heuristic standard deviation for each action class.

The Gaussian model decays too fast around the mean lengths and leads to low accuracies, see Table 8.4. Although the other models perform well, the Poisson distribution still yields the best results.

8.3.4 Impact of Model Components

All three components, the grammar, the length model, and the visual model, contribute their share to restricting the search space to reasonable segmentations. In this section, we evaluate the impact of the grammar and length model on their own and in combination with each other. We use the best-working grammar and length approximation, i.e. the monte-carlo grammar and loss-based mean length approximation, and analyze the effect of omitting the grammar and/or the length model from Equation (8.13) during inference. The results are reported in Table 8.5. Not surprisingly, the performance without a grammar is poor, as the model easily hypothesizes unreasonable action sequences. Adding a grammar alone already boosts the performance, restricting the search space to more reasonable sequences. In order to also get action segments of reasonable length, however, the combination of grammar and length model is crucial. This effect can also be observed in a qualitative segmentation result,

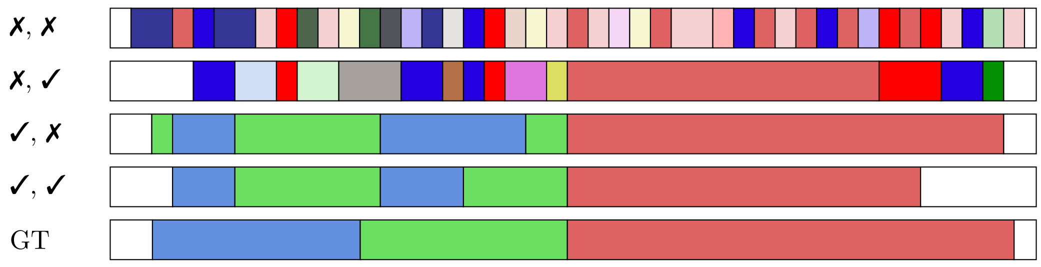


Figure 8.5: Example segmentation on a test video from Breakfast. Row one to four correspond to row one to four from Table 8.5. The last row is the ground truth segmentation.

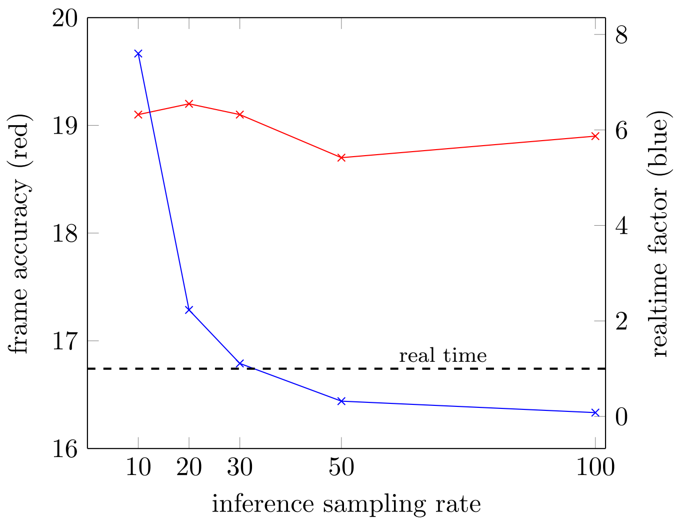


Figure 8.6: Frame accuracy (left axis, red graph) vs. realtime factor of the decoding (right axis, blue graph) for different frame sampling rates on the first split of the Breakfast dataset. Runtime can be greatly reduced by a coarse frame sampling without losing much accuracy.

see Figure 8.5. Note the strong over-segmentation if neither grammar nor length model is used. Introducing the length model partially improves the result but still the grammar is crucial for a reasonable segmentation in terms of correct segment labeling and segment lengths. The fully supervised model (last row of Table 8.5) is trained by assigning the ground truth action label to each video frame. Apart from the labeling, the multi-task network architecture remains unchanged. The full supervision defines an upper bound for the weakly supervised method.

8.3.5 Remarks on the Runtime: Coarse Frame Sampling

As the previous analysis has shown, all three models are rather weak, compared to the previously proposed systems that use more supervision. This raises the question if the Viterbi decoding has to run over all time steps or if a coarse resolution is sufficient. If the models are not accurate on a 10 to 30 frame resolution, Algorithm 7.1 does not need to be applied

	Breakfast	Cooking 2	Holl. Ext.
	frame acc.	*midpoint hit*	*jacc. idx*
Weak supervision: unordered action sets			
monte-carlo	23.3	9.8	9.3
text-based	23.2	10.6	9.2
Stronger supervision: ordered action transcripts			
HMM (Kuehne et al., 2017)	25.9	20.0	8.6
CTC (Huang et al., 2016)	21.8	–	–
ECTC (Huang et al., 2016)	27.7	–	–
HMM/RNN (Chapter 6)	33.3	–	11.9
TCFPN (Ding and Xu, 2018)	38.4	–	12.6
NN-Viterbi (Chapter 7)	43.0	–	12.6

Table 8.6: Performance of the proposed method compared to state-of-the-art methods for weakly supervised temporal segmentation. Note that the proposed method uses action sets as weak supervision, whereas the other approaches rely on stronger supervision with ordered action sequences.

at a fine-grained resolution and the time variable t can run over a coarser grid of frames. In Figure 8.6, the frame accuracy on the first split of Breakfast is compared to the realtime factor of the decoding. A realtime factor of one means video duration and decoding time are equal, a realtime factor smaller than one means the decoding is faster than the video duration. While allowing new segment hypotheses at a rate of 10, 20, or 30 frames results in the best accuracy, coarser sampling rates still allow for similar results, given that the overall model is weak enough that the errors induced by the model outweigh the error induced by the coarse frame sampling. Note that realtime performance is already possible with evaluating every 30th frame in the Viterbi decoding on an Intel Xeon with 2.0 GHz.

8.3.6 Comparison to State-of-the-Art

The task of weakly supervised learning of a model for temporal action segmentation given only action sets has not been addressed before. Still, we can compare the proposed approach to works on temporal action segmentation given ordered action sequences. In this section, we compare the action set approach to the approach of Kuehne et al. (2017), who use hidden Markov models and Gaussian mixture models, as well as the approach from Chapter 7. For completeness, the systems of Huang et al. (2016) and Ding and Xu (2018) are also provided in Table 8.6. All of these approaches use ordered action sequences, and thus a much stronger supervision than the proposed method. Keeping the tremendously large search space for our problem compared to the setting with action transcripts as weak supervision in mind (cf. Section 8.1), our model achieves remarkable results on Breakfast and Hollywood Extended. For a comparison, we also trained the HMM approach of Kuehne et al. (2017) without ordering constraints as supervision. Therefore, we used the same monte-carlo grammar that

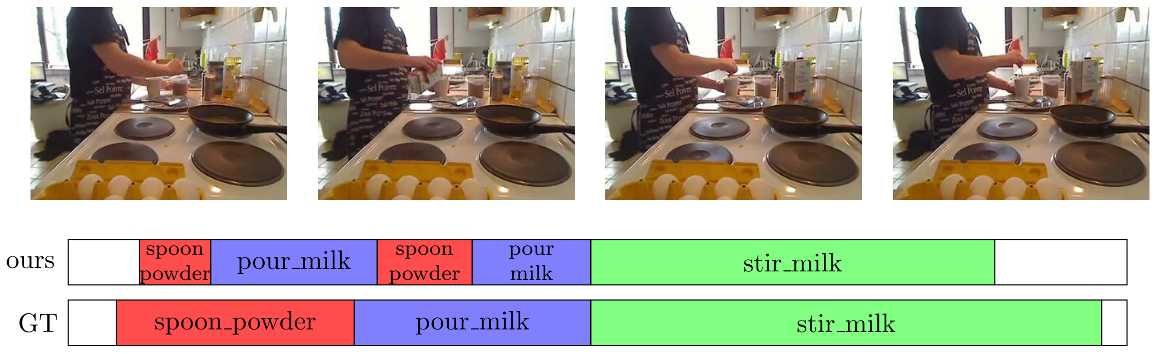


Figure 8.7: Example segmentation. All relevant ground truth actions are present. Note that *spoon_powder* always occurs jointly with *pour_milk*, so it is hard for the model to distinguish them.

cuts per video	4	2	-
avg. #actions per video	12.5	25	50
midpoint hit	17.4	12.1	9.8

Table 8.7: Different levels of video trimming for Cooking 2. More videos and less actions per video result in better performance.

is also used for our action set approach to generate possible ordered action sequences for each training video. These sampled ordered action sequences have then been used for training. This variant of the approach of Kuehne et al. (2017) yields an accuracy of 14.5% on Breakfast, which is far less than our approach. An example segmentation of our approach is shown in Figure 8.7. Falsely recognized actions are frequently those that only occur jointly, such as *spoon_powder* and *pour_milk*. In these cases, the model typically fails to predict the correct ordering.

Impact of Video Length and Amount of Training Data

While the action set approach works well on Breakfast and Hollywood Extended, the results on Cooking 2 show its limitations. The dataset has many classes (67) but only a small amount of training videos (220), which are very long and contain a huge amount of different actions. These characteristics make it difficult for the multi-task learning to distinguish different classes, as many of them occur jointly in most training videos. We show the importance of having enough videos by cutting each video of Cooking 2 into two/four parts (Table 8.7). This increases the number of videos and reduces the number of actions per video. The more videos and the less actions per video on average, the better are the results of our method.

	Breakfast	Cooking 2	Holl. Ext.
	frame acc.	*midpoint hit*	*jacc. idx*
monte-carlo	28.4	10.2	23.0
text-based	28.0	10.6	24.2

Table 8.8: Results of the proposed method when the action sets are provided for inference.

8.3.7 Inference given Action Sets

So far, it has always been assumed that no weak supervision in form of action sets is provided for inference. If the action sets for the videos are, for example, generated using meta-tags of YouTube videos, however, they may as well be available during inference. In this section, we evaluate the method under this assumption.

Let $\mathcal{A}$ be the given action set for a video. During inference, only action sequences that are consistent with $\mathcal{A}$ need to be considered, i.e. for a grammar Γ, only sequences $\mathbf{c}_1^N \in \mathcal{L}(\Gamma) \cap \mathcal{A}^*$ are possible. Since $\mathcal{A}^*$ can be expressed by a right-regular grammar and right-regular grammars are closed under intersection, $\mathbf{c}_1^N \in \mathcal{L}(\Gamma) \cap \mathcal{A}^*$ can again be expressed by a right-regular grammar. If $\mathcal{L}(\Gamma) \cap \mathcal{A}^*$ is empty, we consider all sequences $\mathbf{c}_1^N \in \mathcal{A}^*$. The results are shown in Table 8.8. The above mentioned limitations on Cooking 2 again prevent the method from generating a better segmentation. On Breakfast and Hollywood Extended, a clear improvement of 5% and 15% compared to the inference without given action sets (Table 8.6) can be observed.

In contrast to the results from Table 8.2, for this task the text-based grammar also gives a slight improvement on Hollywood Extended. Still, the improvement is small. On Breakfast, where the textual sources are more off-domain than for Hollywood Extended, there is no improvement at all.

8.4 Summary

We have introduced a system for weakly supervised temporal action segmentation given only unordered action sets. In contrast to ordered action sequences that have been proposed as weak supervision in the previous chapters, action sets are often publicly available in form of meta-tags of videos and do not need to be annotated. Although action sets provide by far less supervision than ordered action sequences and lead to a tremendously large search space, our method still achieves good results. Providing the possibility to incorporate data-driven grammars as well as text-based information or human knowledge, our method can be adapted to specific requirements in different video analysis tasks.

Chapter 9

Conclusion

Contents

9.1 Overview

In this thesis, we addressed the problem of temporal action segmentation with various levels of supervision. The goal of the thesis was twofold: first, advancing the field of action segmentation and detection by proposing a generic framework that is effective and applicable to a wide variety of action modeling tasks and is capable of modeling temporal dependencies of actions on different hierarchical levels. Second, we aimed at providing algorithms that allow for weakly supervised learning and thereby addressed the problem that annotation of vast amounts of video data is expensive and therefore prohibitive for most practical applications.

Current works on temporal action segmentation mainly focus on fully supervised learning and are therefore of limited practical applicability due to annotation cost for settings with diverse video characteristics and many action classes. Moreover, these works typically rely on local decisions and do not take long-range context into account. Our proposed generic framework allows for application of the Viterbi algorithm to find the globally optimal segmentation under the given models. Due to the context model, the temporal extend that influences the recognition of action instances goes beyond local segment dependencies. In Chapter 5, we showed that the framework can be combined with segment-based action classifiers that perform well in classical action recognition and achieved state-of-the-art performance on Thumos. In an attempt to remove the need for full frame-level annotations, we then developed and discussed algorithms for weakly supervised learning from action transcripts and unordered action sets, respectively. Again, the approaches built upon the generic framework. We showed that the interaction of context, length, and visual model improve temporal action segmentation systems and reached state-of-the-art performance on various benchmarks with different levels of supervision.

9.2 Discussion and Contributions

We review the main contributions of this thesis in the following.

Modeling the Long-range Dependencies of Actions

As outlined in Section 1.2.1, humans tend to perceive actions in hierarchical structures. More specifically, a sequence of low-level actions is oftentimes summarized in a high-level concept. We addressed these temporal hierarchies with an explicit decomposition of the segmentation task into a context model, a length model, and a visual model. Using right-regular grammars for the context model, we successfully modeled long-range dependencies between action classes. For the visual model, we mostly relied on RNNs which can model short-term dependencies within an action class. An HMM has been proposed in Chapter 6 as an additional hierarchical element. Modeling subactions as HMM states serves as a further low-level temporal representation. The automatic re-estimation of the number of subactions per action class allows for a decomposition of actions into arbitrarily small fragments.

Note that the quality of the visual model has a strong impact on the need for explicit length modeling. Given a perfect visual model that outputs probability one for the true action class and zero for all other classes at each frame, no length model is required. Imperfect visual models, on the contrary, benefit from explicit length models as they provide a prior on the segment lengths and guide the visual model towards realistic segments. The same holds for the context model. However, even with a perfect visual model, context models can be advantageous in order to distinguish actions that look exactly the same, i.e. that are indistinguishable for the visual model. An example are the actions *stir milk* and *stir coffee* from the Breakfast dataset. Assuming the mug is filmed from the side, the color of the liquid inside the mug is not visible in the frame and both actions can have exactly the same visual appearance. Contextual knowledge, e.g. if milk or coffee has been poured in the mug before, can resolve such ambiguities. An extensive experimental evaluation in fully and weakly supervised settings showed the importance of the high-level context model, the length model, and the visual model.

Developing Algorithms for Weakly Supervised Learning

Addressing the need for algorithms that allow for an analysis of large-scale video data without extensive manual annotation, we proposed different training algorithms ranging from fully supervised to weakly supervised learning. The location of the proposed methods on the supervision graph is shown in Figure 9.1.

Given the availability of high quality speech recognition systems and the fact that there are subtitles for a large amount of videos, action classes can be mined from this information together with a temporal ordering. Even if temporally ordered action transcripts need to be annotated manually, this is much easier than labeling every video frame or exact action boundaries. Action transcripts are therefore an appealing kind of weak supervision. We proposed two effective models and training algorithms in Chapter 6 and Chapter 7 that utilized such annotation and showed considerable improvements over related methods. Closing the

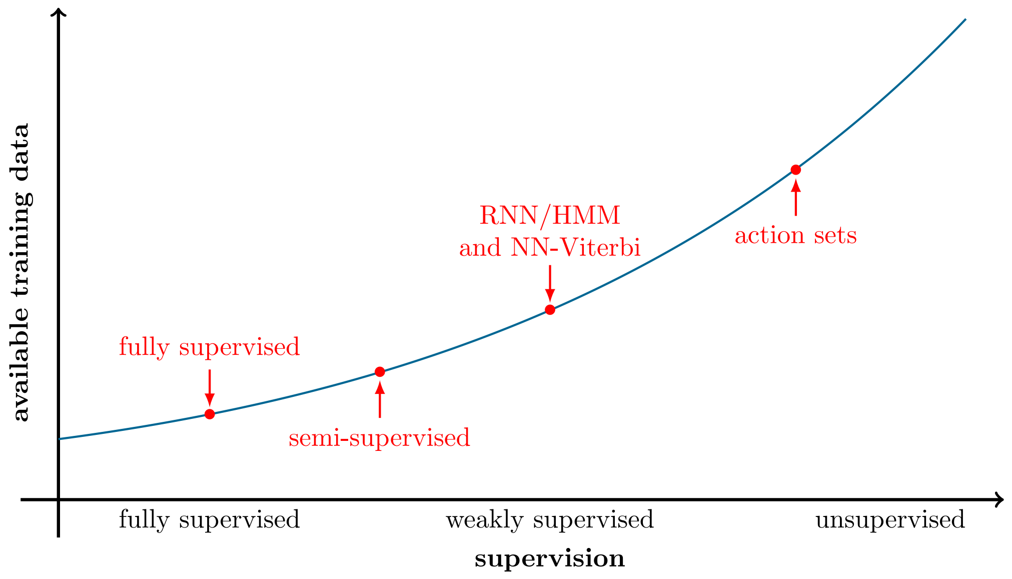


Figure 9.1: Location of the proposed methods on the supervision graph. With the proposed methods, we gradually decreased the required amount of supervision and therefore developed methods that allow to use large amounts of data without tremendous annotation cost.

gap between fully supervised learning and weakly supervised learning, we discussed a semi-supervised setup in Section 7.4. In this setup, sparse frame annotations have been provided in addition to the ordered action transcript. Note that such sparse annotation is easy to obtain by showing human annotators a short clip around the respective frame and asking them which of the classes provided in the action transcript they observed. With only one percent of frame annotations, the approach almost reaches the performance of fully supervised systems. Finally, in Chapter 8, we proposed a method to learn from an unordered set of actions only. This approach allows for a large amount of training data since action sets can be mined from meta-tags or textual descriptions of the video and require very little or no human interference at all.

Unifying Temporal Action Segmentation

Despite considerable improvements on various action detection and segmentation benchmarks over the recent years, existing approaches are usually designed for datasets with specific properties. More precisely, there are two main research streams. One is focusing on datasets like Thumos, where the difficulty is to distinguish multiple instances of (mostly) the same action from background and where the overall background ratio is high, typically above 50%. Methods applied to datasets with such characteristics are closely related to ideas from object detection (Zhao et al., 2017; Chao et al., 2018), i.e. they contain a proposal step to identify possible action segment candidates and a classification step to label these candidates and discard proposals with low scores. The other research stream has a focus on datasets like Breakfast or 50 Salads with a low background ratio but many actions of different classes occurring in each video. Work on datasets with these characteristics is mostly focused on modeling the temporal dependencies of frames within and around an action instance, for example by using temporal convolutional networks with a sufficiently large receptive field (Lea et al., 2016, 2017) or with recurrent neural networks (Huang et al., 2016) or classical temporal models borrowed from automatic speech recognition (Kuehne et al., 2017).

The generic framework proposed in this thesis provides a unification of both research streams. Our fully supervised approaches from Chapter 5 and Section 7.5 have shown state-of-the-art results on Thumos and Breakfast. The explicit length, context, and visual models in the framework depend on the dataset characteristics but the overall approach relied on the same factorization and Viterbi-based decoding scheme.

9.3 Outlook

As discussed in this thesis, algorithms to analyze videos are of importance for many practical applications but the availability of training data is a limiting factor. While the proposed methods for weakly supervised learning are an important step towards being able to gather enough training data, it is desirable to go a step further and learn from video data without the need for any manual interference. Another important research direction is to anticipate the future. When a part of the video has been analyzed and temporally segmented, predicting how the video evolves in the future is of great use to anticipate and plan actions for automated systems that interact with humans. In this section, we briefly discuss limitations of our framework as well as the above-mentioned research directions.

9.3.1 Limitations and Future Improvements

Although the proposed framework and explicit training strategies result in a strong performance on various datasets and in fully and weakly supervised settings, there are some limitations in the current formulation that are worth being addressed in the future.

We could show that the context and length model lead to a significant improvement. However, both components are still constrained to explicit model assumptions. For the length model, we assume a unimodal distribution like a class-dependent Poisson model. The drawback of such models is that they can easily fail on actions with an underlying multimodal length distribution. A more generic approach would be a neural network as length predictor, which is difficult to train given the limited size of current datasets. The context model has similar weaknesses. Using right-regular grammars is a simple approach that is easy to incorporate into the Viterbi decoding. On the other hand, these grammars only recognize regular expressions. Recent advances in language modeling mostly rely on recurrent neural networks (Sundermeyer et al., 2012). Due to the high capacity of language model RNNs, they can represent more complex dependencies and are an interesting candidate for improved context models. Incorporating them into the Viterbi decoding, however, is not trivial.

The training mechanisms used in the proposed methods also allow for further improvements. In the current formulation, the likelihood of the ground truth segmentation is optimized. Discriminative approaches that maximize the margin between the correct segmentation at its competing segmentations are not yet included but have been successful in speech recognition (Bahl et al., 1986; Macherey et al., 2005) and therefore also have great potential for action segmentation. Moreover, in the fully supervised setup, the three model components are optimized completely independently of each other. Future improvements therefore include end-to-end formulations that allow to utilize interdependencies between the three components already during training.

9.3.2 Self-supervised Multi-Modal Learning

Video data frequently comes along with audio. The acoustic information contained in videos, however, is hardly exploited in current systems. Still, audio seems to be a major source of information and allows for self-supervised learning, i.e. learning from the two modalities without the need for any manual annotation at all. The straightforward way to utilize audio is to extract speech using an ASR system and mine actions from this output. In this thesis, we addressed two weakly supervised settings that are a step into the direction of using such data. The approaches proposed in Chapter 6 and Chapter 7 do not rely on exact action boundaries but only need temporal orderings. These orderings are frequently also present in speech. Consider, for instance, instructional videos, where the actions performed in the video signal are simultaneously narrated in the audio signal. For cases in which speech does not follow the same temporal order as the video, action classes mined from the spoken sentences can at least be used as unordered labels, such that the action set approach proposed in Chapter 8 is applicable.

However, using speech data to automatically mine action classes is not trivial. Particularly, there are two major problems that have to be faced. The first is a *relevance problem*. Not everything that is being said is related to the visual signal. Consider, for instance, a soccer match where the sportscaster talks about statistics or other background information that is not directly related to what is happening on the field. The second problem is an *alignment problem*. Speech is not always temporally aligned to the visual events. Again, consider a soccer match. When there is a goal, the sportscaster will typically talk about the goal after it happened. For documentaries, on the contrary, the narrator talks about things that are shown in the video at that particular time. For tutorial videos, the action is frequently described verbally and afterwards demonstrated visually. Thus, in order to utilize audio signals, they need to be warped to the temporal locations in the video they correspond to. Although being a novel research direction, some pioneering works exist. First approaches by Arandjelovic and Zisserman (2017) and Korbar et al. (2018) learn audio-visual correspondences, i.e. a network is trained to decide if an audio signal and a video signal are semantically corresponding. Such methods can be leveraged to address the alignment problem. Further, Chaudhuri et al. (2018) introduce AVA speech, extending the original AVA dataset from Gu et al. (2018) by a labeled audio modality. In this dataset, the audio signal is equipped with four labels indicating speech, speech plus music, speech plus noise, and no speech. The availability of a dataset with annotations for both modalities allows for an in-depth analysis and a better understanding of the relations between video and audio.

9.3.3 Action Anticipation

Many practical applications do not only require to analyze a complete video but to anticipate what is happening in the future. In the field of autonomous driving, for instance, detecting an action early helps to prevent dangerous situations when it comes to interactions between cars and pedestrians. In an early work, Schindler and Van Gool (2008) already show that very short snippets of one to seven frames can lead to good action recognition results. The works of Ma et al. (2016) and Sadegh Aliakbarian et al. (2017) use LSTMs to anticipate actions early after seeing a few frames only. They show good performance on large action recognition benchmarks.

However, it is not only useful to anticipate the immediate future but a larger time horizon, for example to plan actions of an autonomous agent. We address this problem in Abu Farha et al. (2018), where up to five minutes of actions happening in the future are anticipated. The work builds upon our RNN/HMM system from Chapter 6 to generate a temporal segmentation of a smaller portion of already observed video data. Then, an RNN is trained to predict the next action class and its length. The outcome is autoregressively fed back into the network to predict further action segments. As an alternative approach, a CNN based architecture is proposed. The input to the CNN is a one-hot encoding of the ground truth segmentation normalized to a fixed temporal length. The output is a fixed length representation of the predicted future classes. While the RNN in principle can predict an arbitrary duration of future actions, the CNN is limited to a fixed prediction range. An evaluation on Breakfast and 50 Salads shows promising results for both architectures. Given the strong context dependencies in the datasets, both methods can anticipate future actions for up to five minutes with reasonable accuracy.

Despite of its huge potential for practical applications, action anticipation is still in its early stage and needs further attention to develop algorithms that can deal with tremendous uncertainty when predicting the future.

Bibliography

Yazan Abu Farha, Alexander Richard, and Juergen Gall. When will you do what? - Anticipating temporal occurrences of activities. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018. 156

Jean-Baptiste Alayrac, Piotr Bojanowski, Nishant Agrawal, Ivan Laptev, Josef Sivic, and Simon Lacoste-Julien. Unsupervised learning from narrated instruction videos. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 20, 76

Anjum Ali and JK Aggarwal. Segmentation and recognition of continuous human activity. In *IEEE Workshop on Detection and Recognition of Events in Video*, 2001. 15

Humam Alwassel, Fabian Caba Heilbron, Victor Escorcia, and Bernard Ghanem. Diagnosing error in temporal action detectors. In *European Conf. on Computer Vision*, 2018. 3

Relja Arandjelovic and Andrew Zisserman. Look, listen and learn. In *Int. Conf. on Computer Vision*, 2017. 155

Lalit Bahl, Frederick Jelinek, and Robert Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:179–190, 1983. 66

Lalit Bahl, Peter Brown, Peter De Souza, and Robert Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1986. 155

Nicolas Ballas, Li Yao, Pal Chris, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. In *Int. Conf. on Learning Representations*, 2016. 82

Roger Barker and Herbert Wright. *Midwest and its children: The psychological ecology of an American town.* Row, Peterson and Company, 1954. 4

Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. Dynamic image networks for action recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 13

Aaron Bobick and James Davis. Real-time recognition of activity using temporal templates. In *IEEE Winter Conf. on Applications of Computer Vision*, 1996. 15

Aaron Bobick and Yuri Ivanov. Action recognition using probabilistic parsing. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 1998. 19

Piotr Bojanowski, Rémi Lajugie, Francis Bach, Ivan Laptev, Jean Ponce, Cordelia Schmid, and Josef Sivic. Weakly supervised action labeling in videos under ordering constraints. In *European Conf. on Computer Vision*, 2014. xi, 7, 20, 24, 76, 77, 96, 103, 104, 127, 128, 134, 135

Piotr Bojanowski, Rémi Lajugie, Edouard Grave, Francis Bach, Ivan Laptev, Jean Ponce, and Cordelia Schmid. Weakly-supervised alignment of video with text. In *Int. Conf. on Computer Vision*, 2015. 22

Léon Bottou. Online learning and stochastic approximations. In *Online learning and neural networks*. Cambridge University Press, 1998. 111

Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2010. 34

Herve Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*. Springer Science & Business Media, 2012. 113

Matthew Brand, Nuria Oliver, and Alex Pentland. Coupled hidden Markov models for complex action recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 1997. 12

Shyamal Buch, Victor Escorcia, Chuanqi Shen, Bernard Ghanem, and Juan Carlos Niebles. SST: Single-stream temporal action proposals. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 17

Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. ActivityNet: A large-scale video benchmark for human activity understanding. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 16

Fabian Caba Heilbron, Wayner Barrios, Victor Escorcia, and Bernard Ghanem. SCC: Semantic context cascade for efficient action detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 17

Fabian Caba Heilbron, Joon-Young Lee, Hailin Jin, and Bernard Ghanem. What do I annotate next? An empirical study of active learning for action localization. In *European Conf. on Computer Vision*, 2018. 19

Hongping Cai, Fei Yan, and Krystian Mikolajczyk. Learning weights for codebook in image classification and retrieval. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2010. 34

João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 3, 4, 7, 14, 44, 59, 71, 76, 108

Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A Ross, Jia Deng, and Rahul Sukthankar. Rethinking the faster R-CNN architecture for temporal action localization. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018. 3, 17, 47, 56, 72, 73, 74, 131, 153

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014. 44

Sourish Chaudhuri, Joseph Roth, Daniel Ellis, Andrew Gallagher, Liat Kaver, Radhika Marvin, Caroline Pantofaru, Nathan Reale, Loretta Guarino Reid, Kevin Wilson, and Zhonghua Xi. AVA-Speech: A densely labeled dataset of speech activity in movies. In *Interspeech*, 2018. 155

Hsuan-Sheng Chen, Hua-Tsung Chen, Yi-Wen Chen, and Suh-Yin Lee. Human action recognition using star skeleton. In *Proc. of the 4th ACM International Workshop on Video Surveillance and Sensor Networks*, 2006. 12

Yu Cheng, Quanfu Fan, Sharath Pankanti, and Alok Choudhary. Temporal sequence modeling for video event detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014. 19

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conf. on Empirical Methods in Natural Language Processing*, 2014. 37

Xiyang Dai, Bharat Singh, Guyue Zhang, Larry S Davis, and Yan Qiu Chen. Temporal context network for activity localization in videos. In *Int. Conf. on Computer Vision*, 2017. 17

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2005. 33

Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. In *European Conf. on Computer Vision*, 2006. 33

Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conf. on Computer Vision*, 2018. 6

Arthur Danto and Sidney Morgenbesser. What we can do. *The Journal of Philosophy*, 60 (15):435–445, 1963. 4

Arthur C Danto. Basic actions. *American Philosophical Quarterly*, 2(2):141–148, 1965. 4

James Davis and Aaron Bobick. The representation and recognition of human movement using temporal templates. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 1997. 15

Michael Dewar, Chris Wiggins, and Frank Wood. Inference in hidden Markov models with explicit state duration distributions. *IEEE Signal Processing Letters*, 19(4):235–238, 2012. 22

Li Ding and Chenliang Xu. Weakly-supervised action segmentation with iterative soft boundary assignment. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018. xi, 20, 24, 32, 103, 104, 105, 127, 128, 131, 134, 135, 147

Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 76

Olivier Duchenne, Ivan Laptev, Josef Sivic, Francis Bach, and Jean Ponce. Automatic annotation of human actions in video. In *Int. Conf. on Computer Vision*, 2009. 6, 20, 128

Alexei A Efros, Alexander C Berg, Greg Mori, and Jitendra Malik. Recognizing action at a distance. In *Int. Conf. on Computer Vision*, 2003. 12

Eyrun Eyjolfsdottir, Steve Branson, Xavier Burgos-Artizzu, Eric Hoopfer, Jonathan Schor, David Anderson, and Pietro Perona. Detecting social actions of fruit flies. In *European Conf. on Computer Vision*, 2014. 18

Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 14

Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Temporal residual networks for dynamic scene recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017a. 14, 108

Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal multiplier networks for video action recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017b. 14, 44

Chuang Gan, Chen Sun, Lixin Duan, and Boqing Gong. Webly-supervised video recognition by mutually voting for relevant web images and web video frames. In *European Conf. on Computer Vision*, 2016. 22

Jiyang Gao, Zhenheng Yang, Kan Chen, Chen Sun, and Ram Nevatia. Turn tap: Temporal unit regression network for temporal action proposals. In *Int. Conf. on Computer Vision*, 2017a. 16

Jiyang Gao, Zhenheng Yang, and Ram Nevatia. Cascaded boundary regression for temporal action detection. In *British Machine Vision Conference*, 2017b. 16, 73

Rohit Girdhar, Deva Ramanan, Abhinav Gupta, Josef Sivic, and Bryan Russell. ActionVLAD: Learning spatio-temporal aggregation for action classification. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 14

Ross Girshick. Fast R-CNN. In *Int. Conf. on Computer Vision*, 2015. 17

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014. 17

Georgia Gkioxari and Jitendra Malik. Finding action tubes. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 15

Hanlin Goh, Nicolas Thome, Matthieu Cord, and Joo-Hwee Lim. Unsupervised and supervised visual codes with restricted Boltzmann machines. In *European Conf. on Computer Vision*, 2012. 40

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Int. Conf. on Machine Learning*, 2006. 20, 108

Chunhui Gu, Chen Sun, David Ross, Carl Vondrick, Caroline Pantofaru, Yeqing Li, Sudheendra Vijayanarasimhan, George Toderici, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jitendra Malik. AVA: A video dataset of spatio-temporally localized atomic visual actions. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018. 4, 6, 128, 155

Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988. 12

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 14, 44

Georg Heigold, Ralf Schlüter, and Hermann Ney. On the equivalence of Gaussian HMM and Gaussian HMM-like hidden conditional random fields. In *Interspeech*, 2007. 40

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012. 36

Minh Hoai, Zhen-Zhong Lan, and Fernando De la Torre. Joint segmentation and classification of human actions in video. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2011. 18

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997. 37

Rui Hou, Rahul Sukthankar, and Mubarak Shah. Real-time temporal action localization in untrimmed videos by sub-action discovery. In *British Machine Vision Conference*, 2017. 19

De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. Connectionist temporal modeling for weakly supervised action labeling. In *European Conf. on Computer Vision*, 2016. xi, xiii, 20, 24, 32, 76, 96, 103, 104, 113, 119, 127, 128, 129, 130, 134, 135, 147, 154

Haroon Idrees, Amir Zamir, Yu-Gang Jiang, Alex Gorban, Ivan Laptev, Rahul Sukthankar, and Mubarak Shah. The THUMOS challenge on action recognition for videos “in the wild”. *Computer Vision and Image Understanding*, 155:1–23, 2017. 16, 21, 23, 65

Nazlı İkizler and David Forsyth. Searching for complex human activities with no visual examples. *International Journal on Computer Vision*, 80(3):337–357, 2008. 19

Nazlı İkizler, Gokberk Cinbis, and Pinar Duygulu. Human action recognition with line and flow histograms. In *Int. Conf. on Pattern Recognition*, 2008. 12

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Int. Conf. on Machine Learning*, 2015. 114

Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems*, 1999. 34

Mihir Jain, Jan C van Gemert, and Cees G Snoek. What do 15,000 object categories tell us about classifying and localizing actions? In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 13, 24, 56

Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2010. 14

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35 (1):221–231, 2013. 13

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Int. Conf. on Machine Learning*, 2015. 37

Daniel Jurafsky, Chuck Wooters, Jonathan Segal, Andreas Stolcke, Eric Fosler, Gary Tajchaman, and Nelson Morgan. Using a stochastic context-free grammar as a language model for speech recognition. In *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995. 49

Vicky Kalogeiton, Philippe Weinzaepfel, Vittorio Ferrari, and Cordelia Schmid. Action tubelet detector for spatio-temporal action localization. In *Int. Conf. on Computer Vision*, 2017. 15

Svebor Karaman, Lorenzo Seidenari, and Alberto Del Bimbo. Fast saliency based pooling of Fisher encoded dense trajectories. Technical report, University of Florence, 2014. 15, 70, 71

Andrej Karpathy, George Toderici, Sachin Shetty, Tommy Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014. 13, 21, 59

Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017. 3, 21, 23, 44

Yan Ke, Rahul Sukthankar, and Martial Hebert. Efficient visual event detection using volumetric features. In *Int. Conf. on Computer Vision*, 2005. 15

Oscar Koller, Hermann Ney, and Richard Bowden. Deep hand: How to train a CNN on 1 million hand images when your data is continuous and weakly labelled. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 21, 108, 109

Oscar Koller, Sepehr Zargaran, and Hermann Ney. Re-sign: Re-aligned end-to-end sequence modelling with deep recurrent CNN-HMMs. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 21, 108, 109, 122

Bruno Korbar, Du Tran, and Lorenzo Torresani. Cooperative learning of audio and video models from self-supervised synchronization. In *Advances in Neural Information Processing Systems*, 2018. 155

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. 13

Hilde Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. HMDB: A large video database for human motion recognition. In *Int. Conf. on Computer Vision*, 2011. 4, 23

Hilde Kuehne, Ali Arslan, and Thomas Serre. The language of actions: Recovering the syntax and semantics of goal-directed human activities. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014. 4, 5, 19, 23, 77, 104, 105

Hilde Kuehne, Juergen Gall, and Thomas Serre. An end-to-end generative framework for video segmentation and recognition. In *IEEE Winter Conf. on Applications of Computer Vision*, 2016. 19, 36, 96, 104, 105, 131

Hilde Kuehne, Alexander Richard, and Juergen Gall. Weakly supervised learning of actions from transcripts. *Computer Vision and Image Understanding*, 2017. xi, 7, 20, 24, 31, 32, 76, 96, 103, 104, 108, 109, 119, 122, 127, 128, 134, 135, 147, 148, 154

Hilde Kuehne, Alexander Richard, and Juergen Gall. A hybrid RNN-HMM approach for weakly supervised temporal action segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 8

Tian Lan, Yuke Zhu, Amir Roshan Zamir, and Silvio Savarese. Action recognition by hierarchical mid-level action elements. In *Int. Conf. on Computer Vision*, 2015. 15

Ivan Laptev. On space-time interest points. *International Journal on Computer Vision*, 64 (2-3):107–123, 2005. 12

Ivan Laptev and Tony Lindeberg. Space-time interest points. In *Int. Conf. on Computer Vision*, 2003. 12

Ivan Laptev and Patrick Pérez. Retrieving actions in movies. In *Int. Conf. on Computer Vision*, 2007. 13, 15

Ivan Laptev, Marcin Marszalek, Cordelia Schmid, and Benjamin Rozenfeld. Learning realistic human actions from movies. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2008. 6, 20, 33, 76

Colin Lea, Austin Reiter, René Vidal, and Gregory Hager. Segmental spatiotemporal CNNs for fine-grained action segmentation. In *European Conf. on Computer Vision*, 2016. 18, 154

Colin Lea, Michael Flynn, René Vidal, Austin Reiter, and Gregory Hager. Temporal convolutional networks for action segmentation and detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 3, 6, 18, 20, 24, 154

Peng Lei and Sinisa Todorovic. Temporal deformable residual networks for action segmentation in videos. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018. 18

Guy Lev, Gil Sadeh, Benjamin Klein, and Lior Wolf. RNN Fisher vectors for action recognition and image annotation. In *European Conf. on Computer Vision*, 2016. 13

Mengxi Lin, Nakamasa Inoue, and Koichi Shinoda. CTC network with statistical language modeling for action sequence recognition in videos. In *Proceedings of the Thematic Workshops of the ACM Conf. on Multimedia*, 2017a. 20, 127

Tianwei Lin, Xu Zhao, and Zheng Shou. Single shot temporal action detection. In *ACM Conf. on Multimedia*, 2017b. 17

Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. BSN: Boundary sensitive network for temporal action proposal generation. In *European Conf. on Computer Vision*, 2018. 47, 73

David Lowe. Object recognition from local scale-invariant features. In *Int. Conf. on Computer Vision*, 1999. 12

Shugao Ma, Leonid Sigal, and Stan Sclaroff. Learning activity progression in LSTMs for activity detection and early detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 156

Wolfgang Macherey and Hermann Ney. A comparative study on maximum entropy and discriminative training for acoustic modeling in automatic speech recognition. In *Interspeech*, 2003. 40

Wolfgang Macherey, Lars Haferkamp, Ralf Schlüter, and Hermann Ney. Investigations on error minimizing training criteria for discriminative training in automatic speech recognition. In *Interspeech*, 2005. 155

Jonathan Malmaud, Jonathan Huang, Vivek Rathod, Nick Johnston, Andrew Rabinovich, and Kevin Murphy. What's cookin'? Interpreting cooking videos using text, speech and vision. In *Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015. 21

Marcin Marszalek, Ivan Laptev, and Cordelia Schmid. Actions in context. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2009. 20, 24, 76

Osama Masoud and Nikos Papanikolopoulos. A method for human action recognition. *Image and Vision Computing*, 21(8):729–743, 2003. 12

Roland Memisevic and Geoffrey Hinton. Unsupervised learning of image transformations. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2007. 13

Abdel-rahman Mohamed, George Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1): 14–22, 2012. 36

Hiromi Narimatsu and Hiroyuki Kasai. State duration and interval modeling in hidden semi-Markov model for sequential data analysis. *Annals of Mathematics and Artificial Intelligence*, 81(3-4):377–403, 2017. 22

Hermann Ney and Stefan Ortmanns. Dynamic programming search for continuous speech recognition. *IEEE Signal Processing Magazine*, 16(5):64–83, 1999. 49

Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8:1–38, 1994. 57, 58, 66

Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 14, 76

Phuc Nguyen, Ting Liu, Gautam Prasad, and Bohyung Han. Weakly supervised action localization by sparse temporal pooling network. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018. 21

Bingbing Ni, Vignesh Paramathayalan, and Philippe Moulin. Multiple granularity analysis for fine-grained action detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014. 15, 71

Bingbing Ni, Xiaokang Yang, and Shenghua Gao. Progressively parsing interactional objects for fine grained action detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 71

Juan Carlos Niebles and Li Fei-Fei. A hierarchical model of shape and appearance for human action classification. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2007. 13

Dan Oneata, Jakob Verbeek, and Cordelia Schmid. The LEAR submission at Thumos 2014. Technical report, Inria, 2014. 3, 15, 48, 56, 70, 71

Sujoy Paul, Sourya Roy, and Amit Roy-Chowdhury. W-TALC: Weakly-supervised temporal activity localization and classification. In *European Conf. on Computer Vision*, 2018. 21

Xiaojiang Peng, Changqing Zou, Yu Qiao, and Qiang Peng. Action recognition with stacked Fisher vectors. In *European Conf. on Computer Vision*, 2014. 13, 34

Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for image categorization. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2007. 13, 34

Florent Perronnin, Christopher Dance, Gabriela Csurka, and Marco Bressan. Adapted vocabularies for generic visual categorization. In *European Conf. on Computer Vision*, 2006. 34

Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the Fisher kernel for large-scale image classification. In *European Conf. on Computer Vision*, 2010. 35

Hamed Pirsiavash and Deva Ramanan. Parsing videos of actions with segmental grammars. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014. 19

Michael Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*, 1994. 138

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 2015. 17, 72

Alexander Richard and Juergen Gall. A BoW-equivalent recurrent neural network for action recognition. In *British Machine Vision Conference*, 2015. 38

Alexander Richard and Juergen Gall. Temporal action detection using a statistical language model. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 8

Alexander Richard and Juergen Gall. A bag-of-words equivalent recurrent neural network for action recognition. *Computer Vision and Image Understanding*, 156:79–91, 2017. 8, 38

Alexander Richard, Hilde Kuehne, and Juergen Gall. Weakly supervised action learning with RNN based fine-to-coarse modeling. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 8

Alexander Richard, Hilde Kuehne, and Juergen Gall. Action Sets: Weakly supervised action segmentation without ordering constraints. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018a. 9

Alexander Richard, Hilde Kuehne, Ahsan Iqbal, and Juergen Gall. NeuralNetwork-Viterbi: A framework for weakly supervised video learning. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018b. 8

Mikel Rodriguez, Javed Ahmed, and Mubarak Shah. Action MACH: A spatio-temporal maximum average correlation height filter for action recognition. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2008. 12

Marcus Rohrbach, Sikandar Amin, Mykhaylo Andriluka, and Bernt Schiele. A database for fine grained activity detection of cooking activities. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2012. xiii, 3, 5, 15, 25, 47, 56, 65, 71

Marcus Rohrbach, Anna Rohrbach, Michaela Regneri, Sikandar Amin, Mykhaylo Andriluka, Manfred Pinkal, and Bernt Schiele. Recognizing fine-grained and composite activities using hand-centric features and script data. *International Journal on Computer Vision*, 119(3): 346–373, 2016. 25, 143

David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 38

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal on Computer Vision*, 115(3):211–252, 2015. 44

Mohammad Sadegh Aliakbarian, Fatemeh Sadat Saleh, Mathieu Salzmann, Basura Fernando, Lars Petersson, and Lars Andersson. Encouraging LSTMs to anticipate actions very early. In *Int. Conf. on Computer Vision*, 2017. 156

Jorge Sanchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the Fisher vector: Theory and practice. *International Journal on Computer Vision*, 105(3):222–245, December 2013. 35

Sandvine Inc. Global internet phenomena report, 2018. URL `https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf`. 1

Konrad Schindler and Luc Van Gool. Action snippets: How many frames does human action recognition require? In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2008. 156

Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local SVM approach. In *Int. Conf. on Pattern Recognition*, 2004. 3, 12, 33

Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. 37

Fadime Sener and Angela Yao. Unsupervised learning and segmentation of complex activities from video. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2018. 22

Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 1994. 32

Qinfeng Shi, Li Wang, Li Cheng, and Alex Smola. Discriminative human action segmentation and recognition using semi-Markov model. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2008. 18

Zheng Shou, Dongang Wang, and Shih-Fu Chang. Temporal action localization in untrimmed videos via multi-stage CNNs. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 16, 47, 73

Zheng Shou, Jonathan Chan, Alireza Zareian, Kazuyuki Miyazawa, and Shih-Fu Chang. CDC: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 16

Zheng Shou, Hang Gao, Lei Zhang, Kazuyuki Miyazawa, and Shih-Fu Chang. Autoloc: weakly-supervised temporal action localization in untrimmed videos. In *European Conf. on Computer Vision*, 2018. 21

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, 2014. vii, 3, 14, 43, 44, 47, 59, 108

Bharat Singh, Tim Marks, Michael Jones, Oncel Tuzel, and Ming Shao. A multi-stream bi-directional recurrent neural network for fine-grained action detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 16, 76

Gurkirt Singh, Suman Saha, Michael Sapienza, Philip Torr, and Fabio Cuzzolin. Online real-time multiple spatiotemporal action localisation and prediction. In *Int. Conf. on Computer Vision*, 2017. 15

Krishna Kumar Singh and Yong Jae Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *Int. Conf. on Computer Vision*, 2017. 21

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 4, 21, 23

Sebastian Stein and Stephen McKenna. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *ACM Int. Joint Conf. on Pervasive and Ubiquitous Computing*, 2013. 4, 5, 24

Chen Sun, Sanketh Shetty, Rahul Sukthankar, and Ram Nevatia. Temporal localization of fine-grained actions in videos by domain transfer from web images. In *ACM Conf. on Multimedia*, 2015. 22

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Interspeech*, 2012. 154

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 44

Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? In *Int. Conf. on Learning Representations*, 2018. 37, 127

Graham Taylor, Rob Fergus, Yann LeCun, and Christoph Bregler. Convolutional learning of spatio-temporal features. In *European Conf. on Computer Vision*, 2010. 13

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Int. Conf. on Computer Vision*, 2015. 13

Edmondo Trentin and Marco Gori. A survey of hybrid ANN/HMM models for automatic speech recognition. *Neurocomputing*, 37:91–126, 2001. 6

Muhammad Muneeb Ullah, Sobhan Naderi Parizi, and Ivan Laptev. Improving bag-of-features action recognition with non-local cues. In *British Machine Vision Conference*, 2010. 33

Robin Vallacher and Daniel Wegner. What do people think they're doing? Action identification and human behavior. *Psychological Review*, 94(1):3–15, 1987. 4

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. In *ISCA Speech Synthesis Workshop*, 2016. 18

Saeed Vaseghi. State duration modelling in hidden Markov models. *Signal Processing*, 41(1): 31–41, 1995. 22

Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 480–492, 2012. 41, 43

VidCon. Industry Keynote with YouTube CEO Susan Wojcicki, 2015. URL `https://www.youtube.com/watch?v=O6JPxCBlBh8`. 1

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2001. 15

Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. 7, 49

Nam Vo and Aaron Bobick. From stochastic grammar to Bayes network: Probabilistic parsing of complex activity. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014. 19, 109

Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *Int. Conf. on Computer Vision*, 2013. 3, 7, 13, 24, 32, 35, 59, 65

Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Dense trajectories and motion boundary descriptors for action recognition. *International Journal on Computer Vision*, 103(1):60–79, 2013. vii, 13, 14, 32, 33, 34, 35

Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2010. 34

Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition and detection by combining motion and appearance features. Technical report, The Chinese University of Hong Kong and Shenzhen Institutes of Advanced Technology, 2014. 15, 70, 71

Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 14

Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European Conf. on Computer Vision*, 2016. 3, 14, 47, 108

Limin Wang, Yuanjun Xiong, Dahua Lin, and Luc van Gool. Untrimmednets for weakly supervised action recognition and detection. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 21, 48

Daniel Weinland, Remi Ronfard, and Edmond Boyer. A survey of vision-based methods for action representation, segmentation and recognition. *Computer Vision and Image Understanding*, 115:224–241, 2011. 12

Paul Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988. 38

Paul Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78:1550–1560, 1990. 83

Frank Wood, Jan Gasthaus, Cédric Archambeau, Lancelot James, and Yee Whye Teh. The sequence memoizer. *Communications of the ACM*, 54(2):91–98, 2011. 19

Jiajun Wu, Yinan Yu, Chang Huang, and Kai Yu. Deep multiple instance learning for image classification and auto-annotation. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015a. 141

Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *ACM Conf. on Multimedia*, 2015b. 76

Huijuan Xu, Abir Das, and Kate Saenko. R-C3D: region convolutional 3d network for temporal activity detection. In *Int. Conf. on Computer Vision*, 2017. 17, 73

Yan Yan, Chenliang Xu, Dawen Cai, and Jason Corso. Weakly supervised actor-action segmentation via robust multi-task ranking. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 22

Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2009. 34

Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 3, 73

Jun Yuan, Bingbing Ni, Xiaokang Yang, and Ashraf Kassim. Temporal action localization with pyramid of score distribution features. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2016. 16

Zehuan Yuan, Jonathan Stroud, Tong Lu, and Jia Deng. Temporal action localization by structured maximal sums. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2017. 17, 73

Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime TV-L1 optical flow. In *Joint Pattern Recognition Symposium*, 2007. 44

Heiga Zen, Keiichi Tokuda, Takashi Masuko, Takao Kobayasih, and Tadashi Kitamura. A hidden semi-Markov model-based speech synthesis system. *IEICE Transactions on Information and Systems*, 90(5):825–834, 2007. 22

Yue Zhao, Yuanjun Xiong, Limin Wang, Zhirong Wu, Xiaoou Tang, and Dahua Lin. Temporal action detection with structured segment networks. In *Int. Conf. on Computer Vision*, 2017. 6, 16, 48, 72, 73, 74, 131, 153

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit einschlielich Tabellen und Abbildungen –, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen dieser Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. Juergen Gall betreut worden.

Unterschift:
——————————————————————-

Datum:
——————————————————————-