UNIVERSITÄT BONN

Recognizing and Anticipating Human Activities in Videos

DISSERTATION

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

der Mathematisch-Naturwissenschaftlichen Fakultät

der Rheinischen Friedrich–Wilhelms–Universität, Bonn

vorgelegt von

YAZAN ABU FARHA

aus

Al Aqaba, Jordanien

Bonn, 2022

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

1. Gutachter / 1st Advisor: Prof. Dr. Juergen Gall
2. Gutachter / 2nd Advisor: Prof. Dr. Bernt Schiele
Tag der Promotion / Day of Promotion: 21.03.2022
Erscheinungsjahr / Year of Publication: 2022

Abstract

by Yazan Abu Farha

for the degree of

Doctor rerum naturalium

Analyzing human actions in videos has gained increased attention recently. Several approaches, for instance, have been proposed to classify and temporally segment actions in videos. Despite the success of these approaches, they only analyze fully observed videos where all actions in the videos have been carried out. For many applications, however, this is not sufficient and it is crucial to reason beyond what has been observed and anticipate the future. While most existing methods for anticipation predict only the very near future, making long-term predictions over more than just a few seconds is a task with many practical applications that has been overlooked in the literature. In this thesis, we therefore propose approaches that predict a considerably large number of future actions and their durations for up to several minutes into the future. Nonetheless, the ability to predict the future depends on the level of understanding of what has been observed. As recent approaches for analyzing activities in long untrimmed videos either suffer from over-segmentation errors or cannot be trained end-to-end, we first propose approaches to temporally segment activities in long videos to address the limitations of the previous approaches. Then, we combine the proposed segmentation and the anticipation approaches in an end-to-end framework for long-term anticipation of activities.

To this end, we first propose two multi-stage architectures based on temporal convolutional networks for the temporal action segmentation task. We further introduce a novel smoothing loss that penalizes over-segmentation errors and improves the quality of the predictions. While the proposed segmentation models show a strong performance in recognizing and segmenting activities, they depend on the availability of large fully annotated datasets. Annotating such datasets with the start and end time of each action segment is, however, very costly and time-consuming. Although several approaches have been proposed to reduce the level of required annotations to only an ordered list of the occurring actions in the video, the performance of these approaches is still much worse than fully supervised approaches. Therefore, we propose to use a new level of supervision based on timestamps, where only a single frame is annotated for each action segment. We demonstrate that models trained with timestamp supervision achieve comparable performance to fully supervised approaches using a tiny fraction of the annotation cost.

For the long-term anticipation of activities, we propose a two-step approach that first infers the actions in the observed frames using an action segmentation model, then the inferred actions are used to predict the future actions. As predicting longer into the future requires considering the uncertainty in the future actions, we further propose a framework that predicts a distribution over the future action segments and use it to sample multiple possible sequences of future actions.

To further improve the anticipation performance, we combine the proposed segmentation

and the anticipation models in an end-to-end framework. Furthermore, as predicting the future from the past and the past from the future should be consistent, we introduce a cycle consistency loss over time by predicting the past activities given the predicted future. The end-to-end framework outperforms the two-step approach by a large margin.

For both the action segmentation and anticipation tasks, we evaluate the proposed approaches on several public datasets with long videos containing many action segments. Extensive evaluation shows the effectiveness of the proposed approaches in capturing long-range dependencies and generating accurate predictions.

Keywords: temporal action segmentation, temporal convolutional network, long-term anticipation, activity forecasting

Acknowledgements

I would like to thank my advisor Prof. Juergen Gall for all the guidance and support. I was privileged during my PhD and had the freedom to work on my own ideas and yet he always provided a thorough feedback and many suggestions to polish these ideas and improve the quality of the work. I have been always amazed by the amount of effort and time he puts until the last moment before the deadlines to refine our submissions.

Furthermore, I would like to thank the dissertation committee members: Prof. Bernt Schiele, Prof. Maren Bennewitz, and Prof. Alexander Effland for taking the time to review the thesis.

I'm also grateful to my colleagues and friends in the Computer Vision Group at University of Bonn. Their support, kindness, and insightful discussions made my stay at Bonn a pleasant experience. I would like also thank my collaborators for the fruitful collaboration.

Additionally, I want to thank the German Academic Exchange Service (DAAD) for supporting my master's studies at the University of Bonn. Their support was a main factor for advancing my academic career and then deciding to extend my stay in Germany and pursue a PhD at the same university.

Finally, many thanks to my parents, sister, and brother for their unconditional love and support.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

An alphabetically sorted list of abbreviations used in the thesis:

BPTT	Back-Propagation Through Time
CNN	Convolutional Neural Network
DDL	Dual Dilated Layer
EOS	End-Of-Sequence
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Network
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
HOF	Histograms of Optical Flow
HOG	Histograms of Oriented Gradients
IDT	Improved Dense Trajectories
IoU	Intersection over Union
LSTM	Long Short-Term Memory
MBH	Motion Boundary Histograms
MLP	Multi-Layer perceptron
MoC	Mean over Classes
MSE	Mean Squared Error
MS-TCN	Multi-Stage Temporal Convolutional Network
PCA	Principal Component Analysis
RBF	Radial Basis Function
RNN	Recurrent Neural Network
SOS	Start-Of-Sequence
SS-TCN	Single-Stage Temporal Convolutional Network
SVM	Support Vector Machine
TCN	Temporal Convolutional Network
TN	True Negative
TP	True Positive
TSN	Temporal Segment Network

List of Publications

The thesis is based on the following publications:

- **When will you do what? - Anticipating Temporal Occurrences of Activities**
  Yazan Abu Farha, Alexander Richard, and Juergen Gall
  IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

- **MS-TCN: Multi-Stage Temporal Convolutional Network for Action Segmentation**
  Yazan Abu Farha and Juergen Gall
  IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

- **Uncertainty-Aware Anticipation of Activities**
  Yazan Abu Farha and Juergen Gall
  IEEE International Conference on Computer Vision Workshops, 2019.

- **Long-Term Anticipation of Activities with Cycle Consistency**
  Yazan Abu Farha, Qiuhong Ke, Bernt Schiele, and Juergen Gall
  German Conference on Pattern Recognition (GCPR), 2020.

- **MS-TCN++: Multi-Stage Temporal Convolutional Network for Action Segmentation**
  Shijie Li*, Yazan Abu Farha*, Yun Liu, Ming-Ming Cheng, and Juergen Gall
  IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020.
  (* denotes equal contribution)

- **Temporal Action Segmentation from Timestamp Supervision**
  Zhe Li, Yazan Abu Farha, and Juergen Gall
  IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

CHAPTER 1

Introduction

Contents

1.1 Motivation

Consider the video snippets illustrated in Figure 1.1. Can you predict how the video would evolve into the future? It is not surprising that you can predict that the cook would add topping on the pizza, bake it, and then slice and serve it. In fact, we are continuously predicting the future and it is hard to imagine living without such ability. Imagine yourself, for instance, in a busy Monday morning trying to cross the street. Even for such simple task, you would probably look around, identify moving objects, and predict how they would move to avoid colliding with them. Almost every aspect of our life relies on our ability to predict the future. When you help a friend to carry a new table upstairs you continuously predict your friend's actions and moves to ensure smooth interactions. During a penalty kick in a football game, the goalkeeper has to anticipate the direction of the ball even before the opposing player approaches it. Similarly, a chess player has to predict the movements of his opponent several steps into the future to decide on his next move. Furthermore, future prediction plays a significant role in many aspects of our societies such as weather and stock market forecasts, navigation systems, decision making, and strategic planning.

As future prediction is in every aspect of our lives, it is crucial to model such capability in intelligent agents. Nonetheless, the question is what to predict. Since we perceive the world as a sequence of frames, initial approaches tried to predict future frames in video sequences at pixel level (*Ranzato et al.*, 2014; *Mathieu et al.*, 2016). However, predicting in such high dimensional space is challenging and these approaches failed at generating realistic predictions, which limits

Figure 1.1: Look at this sequence of video frames. Can you guess what are the next steps? While such predictive task seems trivial for humans, it is still challenging for computer systems. In this thesis, we address this task and propose multiple models that are capable of predicting the sequence of future activities and their durations. [1]

their applications. While other approaches tried to predict sentences describing future frames (*Sener and Yao*, 2019; *Mahmud et al.*, 2021), such representation cannot be directly used for real-world applications.

As a middle ground, predicting future activities in video sequences has emerged (*Lan et al.*, 2014; *Vondrick et al.*, 2016; *Gao et al.*, 2017c; *Shi et al.*, 2018). Despite the success of these approaches, they are limited to predicting the next action just a few seconds into the future. For many real world applications, however, making long-term predictions beyond just a few seconds is crucial. Therefore, the goal of this thesis is to extend the capabilities of intelligent systems to predict longer time horizons that span several minutes into the future. Nonetheless, predicting longer time horizons requires better understanding of the observed frames and capturing temporal dependencies over very long video sequences. To this end, we also address the action segmentation task, which temporally segments actions in long videos, and use the recognized actions to predict the future.

Recognizing and anticipating human activities in videos has many applications. In robotics, for instance, the ability to predict the future activities results in better human-robot interactions (*Koppula and Saxena*, 2016). Furthermore, service robots that predict the future can utilize this information to decide where to place themselves to provide a timely service for users (*Bruckschen et al.*, 2020). The ability to recognize and forecast activities can also be used in assembly lines to detect mistakes or missing important steps (*Soran et al.*, 2015). A new application has also emerged in the medical domain, especially in the computer-assisted surgery (*Rivoir et al.*, 2020) and surgical skill assessment (*Liu et al.*, 2021). Moreover, the techniques discussed throughout this thesis can be useful for many other applications such as animal behaviour analysis, anomaly detection (*Öztürk and Can*, 2021), and sign language segmentation (*Renz et al.*, 2021).

In this thesis, we contribute to both the action segmentation and the anticipation tasks. We propose approaches that are capable of temporally parsing long videos into the different action segments that are depicted in the videos. We further introduce approaches that can anticipate sequences of possible future activities that are likely to happen after observing partial video sequences. Finally, we combine both approaches in a unified end-to-end framework for long-term anticipation.

[1] Video source: `https://tasty.co/`.

1.2 Problem Statement

The proposed approaches in this thesis fall under either one of two video understanding tasks: temporal action segmentation, and long-term anticipation of activities. On the one hand, the action segmentation task predicts an action label for each frame in the input video. On the other hand, the anticipation task reasons beyond the observed frames and predicts a sequence of future activities with their durations. In this section, we provide a formal definition for both tasks.

1.2.1 Temporal Action Segmentation

The temporal action segmentation task can be defined by describing the expected input and the desired output for that input. The input for this task is a long untrimmed video that contains multiple actions. Let us denote the input video by $x_{1:T} = (x_1, \ldots, x_T)$, where T is the number of frames and x_t is the frame at time step t, represented by either an RGB image or as a feature vector representation. The goal of the temporal action segmentation task is to predict an action label for each frame in the input video x. *I.e.* the output is a sequence $a_{1:T} = (a_1, \ldots, a_T)$ of length T, where a_t is the action label for the frame x_t. We assume that the action labels a_t are coming from a pre-defined set of C action classes.

The sequence of the frame-wise action labels $a_{1:T}$ can also be represented in a segment-wise form. By grouping consecutive frames with the same action label into a single segment, the frame-wise action labels $a_{1:T}$ can be represented as sequence of action segments $A_{1:N} = (A_1, \ldots, A_N)$ with their corresponding durations $\ell_{1:N} = (\ell_1, \ldots, \ell_N)$, where N is the total number of segments in the sequence of T frames and $\sum \ell_i = T$. In this representation, A_i corresponds to the action label for the $i\text{-}th$ segment and ℓ_i is an integer number that corresponds to the number of frames belonging to that segment.

1.2.2 Long-Term Anticipation of Activities

The goal of the long-term action anticipation is to predict a sequence of future activities with their durations form a partially observed video. Formally, let $x_{1:T} = (x_1, \ldots, x_T)$ be a video with T frames. Given the first t frames $x_{1:t}$, the task is to predict the actions happening from frame $t+1$ to T. That is, we aim to assign action labels $a_{t+1:T} = (a_{t+1}, \ldots, a_T)$ to each of the unobserved frames. Using the equivalent segment-wise representation, the observed video sequence can also be represented as n action segments $A_{1:n} = (A_1, \ldots, A_n)$ with their corresponding lengths $\ell_{1:n} = (\ell_1, \ldots, \ell_n)$. And the target are the segments $A_{n+1:N}$ with the corresponding segments lengths $\ell_{n+1:N}$, where N is the total number of action segments in the video. This novel task is one of the contributions of this thesis, which is published in (*Abu Farha et al.*, 2018) and will be discussed in Chapter 6.

1.3 Challenges

Analyzing long untrimmed videos with multiple actions imposes many challenges. In this section, we discuss some of these challenges and open questions that need to be addressed for building models that can segment and anticipate human activities.

Long-Range Dependencies

Since a video is a time series of frames, frames sampled from the same video are not independent. What happens at the beginning of the video would very likely affect what will happen at the end of that video. Therefore, naively treating each frame independently without its temporal context is sup-optimal. Modelling such dependencies between frames allows for better understanding of the video content, especially for ambiguous frames due to occlusions or motion blur. Furthermore, reasoning over long-range temporal context would enable models to extrapolate and predict what is likely to happen in the future.

Costly Annotations

The success of deep learning methods relies on the availability of large annotated datasets. Nonetheless, when it comes to video understanding, the annotation process becomes very costly and cumbersome. For instance, for the temporal action segmentation task, videos are several minutes long and span thousands of frames. Annotating every single frame in large collections of such videos would require a significant amount of resources. Furthermore, different annotators might disagree on what defines an action boundary and when the transition happens from one action segment to the next one. Therefore, approaches that can be trained with less supervision are of great importance for the video understanding domain.

Good Representations for Anticipation

Anticipating future activities form a partially observed video might seem easy at first glance. Nonetheless, as videos consist of long sequences of frames, even a partially observed video can still span several thousands of frames. Learning from such very high dimensional space is very challenging. Therefore, it is crucial to learn representations that capture the content of the videos in a low dimensional space.

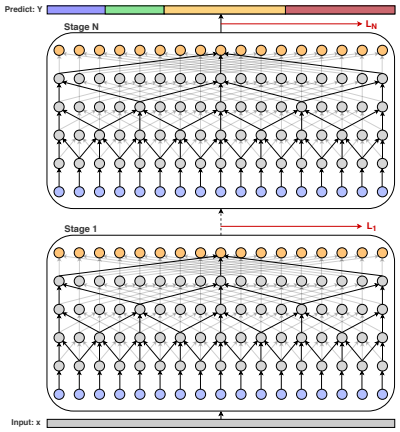
Modelling Uncertainty

Making predictions about the future is challenging. The future is inherently uncertain. For the same observed sequence of activities, there are many possible ways in which the future can evolve. Thus, only predicting the most likely future is insufficient for many practical applications and it is crucial to model the uncertainty over future activities.

1.4 Contributions

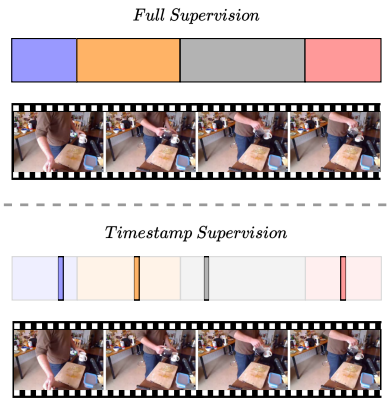
In this thesis, we propose approaches to address the challenges discussed above. First, we propose a class of models for recognizing and temporally segmenting action segments in long untrimmed videos. Then, we propose methods to anticipate sequences of future activities with their durations from partially observed videos. Finally, we combine these approaches in a single framework for long-term anticipation. Our contributions can be summarized as follows.

1.4.1 Multi-Stage Architectures for Temporal Action Segmentation



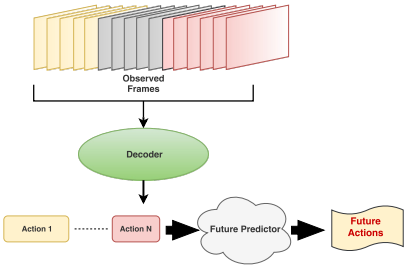
Our first contribution is two multi-stage architectures for the temporal action segmentation task. In both architectures, we compose several models sequentially such that each model operates directly on the output of the previous one, which results in an incremental refinement of the predictions. To capture long-range dependencies, each stage is constructed by stacking several layers of dilated temporal convolutions. Using dilated convolutions increases the receptive field of each stage without significantly increasing the number of parameters. While the first architecture has small receptive fields at shallow layers and gradually increases it by adding more layers, the second architecture combines both large and small receptive fields at each layer. To train the proposed architectures, we further introduce a smoothing loss that penalizes over-segmentation errors. We demonstrate the effectiveness of our models in capturing long-range dependencies between action classes and producing high quality predictions on multiple action segmentation benchmarks. Furthermore, the proposed models are view-agnostic and work well on videos captured from different view-points including top and egocentric views.

1.4.2 Timestamp Supervision for Action Segmentation



Although the proposed multi-stage architectures discussed above have been very successful in segmenting actions, training these models requires fully annotated videos with the precise start and end time of each action segment. As videos are composed of thousands of frames, annotating every single frame is a very costly and time-consuming process. While recent approaches showed that it is feasible to train an action segmentation model using only an ordered list of actions that occur in the training videos, the performance of these approaches is still significantly worse than the performance of the fully supervised approaches. As a second contribution, we propose to use a new level of supervision that addresses the limitations of the previous approaches. To this end, we propose to use timestamp supervision for the temporal action segmentation task by only annotating a single frame from each action segment in the training videos. Timestamps require a comparable annotation effort to annotating the list of actions occurring in the video, and yet provide a more supervisory signal. We demonstrate the effectiveness of timestamp supervision by proposing the first action segmentation approach that is trained using only timestamps annotations. Our approach uses the model output and the annotated timestamps to generate frame-wise labels by detecting the action changes. We further introduce a novel confidence loss that forces the predicted probabilities to monotonically decrease as the distance to the timestamps increases. We show that models trained with timestamps annotations achieve comparable performance to the fully supervised approaches at a tiny fraction of the annotation costs.

1.4.3 Long-Term Anticipation of Activities

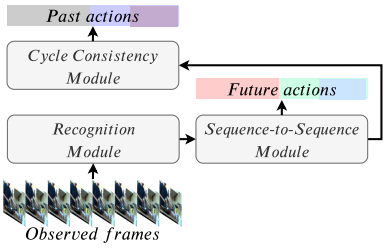


Our contributions so far only address analyzing human actions in fully observed videos. However, for many applications it is beneficial to reason beyond what has happened and predict what will happen in the future. State-of-the-art approaches addressing this task are, however, limited to anticipating only the immediate future. As a third contribution, we therefore propose to extend the anticipation task and make long-term predictions over more than just a few seconds. We propose two methods to predict a sequence of future actions and their durations, which can take up to five minutes into the future. Both models are trained in a two-step approach. In the first step, we train an off-the-shelf action segmentation model that predicts the action segments in the observed frames. Then, we use the proposed models to predict the future actions from the observed action segments. We show that our methods generate accurate predictions of the future even for long videos with a huge amount of different actions and can even deal with noisy or erroneous input information.

1.4.4 Uncertainty-Aware Framework for Anticipating Activities

One of the contributions of this thesis is the introduction of the long-term anticipation task, which extends the prediction time horizon to several minutes into the future. However, as increasing the predicted time horizon, the future becomes more uncertain and models that generate a single prediction fail at capturing the different possible future activities. Therefore, as a fourth contribution, we extend the approach discussed in the previous section to predict multiple possible sequences of future activities. Both an action model and a length model are trained to predict the probability distribution of the future activity and its duration. At test time, we sample from the predicted distributions multiple samples that correspond to the different possible sequences of future activities. Evaluation on two benchmarks shows that our approach is able to capture the multi-modal nature of future activities without compromising the accuracy when only predicting a single sequence of activities.

1.4.5 End-to-End Framework for Long-Term Anticipation of Activities



Despite the success of our methods in predicting activities for a longer time horizon into the future, they are trained in a two-step approach. While such a two-step approach demonstrates that action labels are good representations to predict the future, it decouples the semantic interpretation of the observed sequence from the anticipation task. As a last contribution, we propose a framework for anticipating future activities directly from the features of the observed frames and train it in an end-to-end fashion. Building on the success of the two-step approach, we cast the recognition

of the observed frames as an auxiliary task that encourages learning discriminative representations. Furthermore, following the success of cycle consistency in computer vision, we show that the anticipation performance can be further improved by forcing the model to predict the past activities given the predicted future.

1.5 Thesis Structure

The rest of this thesis is organized as follows:

Chapter 2 provides an overview of the related work on action recognition, temporal action segmentation, action anticipation, and anticipation of other representations. It also provides a brief summary of the datasets that are used to evaluate the proposed approaches.

Chapter 3 gives a brief overview of the concepts that are used throughout the thesis. It starts with a quick introduction to neural networks, including convolutional and recurrent networks. Then, it describes the video representations that are used as input to the proposed approaches. Finally, it provides a formal definition of the evaluation metrics that are used to evaluate the performance of the proposed approaches.

Chapter 4 proposes two multi-stage temporal convolutional networks for the temporal action segmentation task. It also proposes a novel smoothing loss to reduce over-segmentation errors. This chapter is based on (*Abu Farha and Gall*, 2019a; *Li et al.*, 2020).

Chapter 5 introduces timestamp supervision for the temporal action segmentation task. In contrast to the full supervision, timestamp supervision only annotates one frame from each action segment in the training videos. The chapter further proposes a framework to train an action segmentation model using timestamps annotations. This chapter is based on (*Li et al.*, 2021).

Chapter 6 proposes the novel task of long-term anticipation of activities. While state-of-the-art approaches only anticipate a single activity a few seconds into the future, this chapter proposes two approaches that anticipate future activities with their durations up to five minutes into the future. This chapter is based on (*Abu Farha et al.*, 2018).

Chapter 7 extends the approach proposed in **Chapter 6** to predict multiple possible sequences of future activities. This chapter is based on (*Abu Farha and Gall*, 2019b).

Chapter 8 proposes an end-to-end framework for long-term anticipation of activities. It further introduces a cycle consistency loss over time by forcing the model to predict the past activities from the predicted future. This chapter is based on (*Abu Farha et al.*, 2020).

Finally, conclusions are given in **Chapter 9** along with suggested directions for future work.

CHAPTER 2

Related Work

In this chapter, we discuss the related work for this thesis. First, we discuss approaches for action recognition which classify trimmed videos. Second, we review action segmentation approaches that handle long untrimmed videos. Third, we provide an overview of the methods that anticipate the future with more focus on action anticipation. Finally, we briefly discuss the datasets that are used throughout the thesis.

Contents

2.1 Action Recognition

The goal of action recognition is to classify a short video clip into a pre-defined action class. The input videos, however, have to be pre-segmented and depict only a single action class. While in real world scenarios videos are untrimmed and contain multiple activities spanning long durations, the ability to recognize short clips is still useful in advancing research on analyzing long videos. Action recognition approaches can learn strong representations that can be used for many video understanding tasks. Therefore, we provide a brief overview of the advances in the action recognition research.

Traditional action recognition approaches relied on hand-crafted features such as space-time interest points (*Laptev*, 2005) and dense trajectories (*Wang et al.*, 2011, 2013). In particular the improved dense trajectories (*Wang and Schmid*, 2013) combined with the Fisher vectors encoding (*Sánchez et al.*, 2013) showed a strong performance on different action recognition benchmarks,

and even outperformed early attempts for applying convolutional neural networks (CNNs) on the action recognition task (*Karpathy et al.*, 2014).

Recent approaches for action recognition are deep learning based approaches and can be divided into two groups: approaches that follow the two-stream architecture proposed by *Simonyan and Zisserman* (2014), and approaches that are based on 3D convolutional neural networks (3D CNNs). The two-stream network of *Simonyan and Zisserman* (2014) consists of two parallel 2D CNNs: a spatial CNN and a temporal CNN. The spatial network performs action recognition from a single video frame. Whereas the temporal network is trained to recognize actions from stacks of optical flow frames. The final prediction is obtained by aggregating the predictions from both streams. Several approaches built on top of the two-stream architecture and further improved its performance. For instance, *Feichtenhofer et al.* (2016) used better network architecture based on ResNet (*He et al.*, 2016) and further introduced residual connections between the two streams. To capture long-range temporal structure, *Wang et al.* (2016) introduced the temporal segment network (TSN). TSN divides the input video into multiple segments and sparsely samples snippets from each segment to utilize the visual information in the entire video. The final prediction is obtained by aggregating the predictions from all snippets. Similarly, *Zhou et al.* (2018) sample frames at different temporal scales to capture temporal relations. In (*Diba et al.*, 2017), a temporal linear encoding layer was proposed to aggregate information throughout the entire video.

Another class of action recognition approaches utilized spatio-temporal convolutions, which are also known as 3D CNNs. While initial attempts of applying 3D CNNs for action recognition performed worse than hand-crafted features (*Tran et al.*, 2015), it was until the introduction of the I3D model (*Carreira and Zisserman*, 2017) when 3D CNNs significantly outperformed other approaches and took action recognition to the next level. The I3D network is designed by inflating successful 2D CNN architectures into 3D and using its pre-trained weights on the ImageNet dataset (*Deng et al.*, 2009) for initializing the 3D CNN weights. Following the success of I3D, several approaches have been proposed and advanced the capabilities of 3D CNNs in recognizing activities. In (*Qiu et al.*, 2017; *Tran et al.*, 2018) the 3D convolutions are factorized into 2D spatial convolutions and 1D temporal convolutions, which reduced the complexity of training the network. *Diba et al.* (2018) introduced a spatio-temporal channel correlation block that can be integrated with 3D CNNs to model the correlations between channels. Other approaches focused on modelling long range dependencies both in the temporal and spatial domains (*Varol et al.*, 2018; *Wang et al.*, 2018; *Wu et al.*, 2019). In (*Feichtenhofer et al.*, 2019), a 3D CNN with two pathways was proposed. The first pathway operates on a low frame rate and captures spatial semantics. Whereas the second pathway operates on a high frame rate and captures motion cues. In (*Feichtenhofer*, 2020), an efficient network has been proposed by progressively expanding a tiny 2D image classification architecture into 3D. Recent approaches improved the efficiency of action recognition models by learning an adaptive spatial resolution (*Meng et al.*, 2020), adaptive temporal resolution (*Fayyaz et al.*, 2021), or by utilizing the temporal differences between features at different levels of the network to capture multi-scale temporal information (*Wang et al.*, 2021).

2.2 Temporal Action Segmentation

In the previous section, we discussed advances in recognizing activities in short trimmed videos. In this section, we discuss approaches that analyze long untrimmed videos with many activities.

Depending on the level of supervision, we divide these approaches into two groups: fully supervised approaches and weakly supervised approaches.

2.2.1 Fully Supervised Approaches

Temporal action segmentation has received a lot of interest from the computer vision community. Many approaches were proposed to localize action segments in videos or assign action labels to video frames. In earlier approaches, a sliding window approach is applied with non-maximum suppression (*Rohrbach et al.*, 2012; *Karaman et al.*, 2014). However, such approaches are computationally expensive since the model has to be evaluated at different window scales. Other approaches model actions based on the change in the state of objects and materials (*Fathi and Rehg*, 2013) or based on the interactions between hands and objects (*Fathi et al.*, 2011a). *Bhattacharya et al.* (2014) use a vector time series representation of videos to model the temporal dynamics of complex actions using methods from linear dynamical systems theory. The representation is based on the output of pre-trained concept detectors applied on overlapping temporal windows. *Cheng et al.* (2014) represent videos as a sequence of visual words, and model the temporal dependency by employing a Bayesian non-parametric model of discrete sequences to jointly classify and segment video sequences.

Despite the success of the previous approaches, their performance was limited as they failed in capturing the context over long video sequences. To alleviate this problem, many proposals tried to employ high level temporal modeling over frame-wise classifiers. *Kuehne et al.* (2016) represent the frames of a video using Fisher vectors of improved dense trajectories, and then each action is modeled with a hidden Markov model (HMM). These HMMs are combined with a context-free grammar for recognition to determine the most probable sequence of actions. HMMs are also used in many other approaches. *Kuehne et al.* (2017) combine HMMs with a Gaussian mixture model (GMM) as a frame-wise classifier. However, since frame-wise classifiers do not capture enough context to detect action classes, *Richard et al.* (2017) and *Kuehne et al.* (2020) use a GRU instead of the GMM that is used in (*Kuehne et al.*, 2017). A hidden Markov model is also used in (*Tang et al.*, 2012) to model both transitions between states and their durations. *Vo and Bobick* (2014) use a Bayes network to segment activities. They represent compositions of actions using a stochastic context-free grammar with AND-OR operations. *Richard and Gall* (2016) propose a model for temporal action detection that consists of three components: an action model that maps features extracted from the video frames into action probabilities, a language model that describes the probability of actions at sequence level, and finally a length model that models the length of different action segments. To get the video segmentation, they use dynamic programming to find the solution that maximizes the joint probability of the three models. *Singh et al.* (2016a) use a two-stream network to learn representations of short video chunks. These representations are then passed to a bi-directional LSTM to capture dependencies between different chunks. However, their approach is very slow due to the sequential prediction. In (*Singh et al.*, 2016b), a three-stream architecture that operates on spatial, temporal and egocentric streams is introduced to learn egocentric-specific features. These features are then classified using a multi-class SVM.

Inspired by the success of temporal convolution in speech synthesis (*Van Den Oord et al.*, 2016), researchers have tried to use similar ideas for the temporal action segmentation task. *Lea et al.* (2017) propose a temporal convolutional network for action segmentation and detection. Their approach follows an encoder-decoder architecture with a temporal convolution and pooling in the encoder,

and upsampling followed by deconvolution in the decoder. While using temporal pooling enables the model to capture long-range dependencies, it might result in a loss of fine-grained information that is necessary for fine-grained recognition. *Lei and Todorovic* (2018) build on top of (*Lea et al.*, 2017) and use deformable convolutions instead of the normal convolution and add a residual stream to the encoder-decoder model. *Ding and Xu* (2018) add lateral connections to the encoder-decoder TCN (*Lea et al.*, 2017) and propose a temporal convolutional feature pyramid network for predicting frame-wise action labels. Recently, *Mac et al.* (2019) propose to learn spatio-temporal features using deformable convolutions and local consistency constraints. All of the approaches in (*Lea et al.*, 2017; *Lei and Todorovic*, 2018; *Ding and Xu*, 2018), however, operate on downsampled videos with a temporal resolution of 1-3 frames per second. In our works (*Abu Farha and Gall*, 2019a; *Li et al.*, 2020) we proposed two multi-stage architectures that operate on the full temporal resolution of the videos, which will be discussed in Chapter 4. Recent approaches built on top of our architectures and further refined the predictions using graph convolutional networks (*Huang et al.*, 2020), boundary-aware pooling (*Wang et al.*, 2020; *Ishikawa et al.*, 2021), or by using neural networks architecture search to optimize the receptive field of the model by adapting the dilation factors in each layer (*Gao et al.*, 2021).

Action detection is a related but a different task. In this context, the goal is to detect sparse action segments while most parts of the videos are unlabeled. In this work, we focus on action segmentation where the videos are densely annotated. For action detection, several approaches follow a two stage pipeline. The first stage generates proposals, whereas the second stage classifies and refines the boundaries of these proposals (*Shou et al.*, 2016; *Xu et al.*, 2017; *Zhao et al.*, 2017; *Chao et al.*, 2018; *Zeng et al.*, 2019; *Gao et al.*, 2017a; *Buch et al.*, 2017b; *Gao et al.*, 2017b; *Lin et al.*, 2019). Other approaches combine the proposal generation and classification in a single-stage architecture which enables end-to-end training (*Yeung et al.*, 2016; *Buch et al.*, 2017a; *Long et al.*, 2019; *Lin et al.*, 2021).

2.2.2 Weakly Supervised Approaches

The approaches discussed in the previous section require training videos that are annotated with frame-wise labels. However, densely annotating all the video frames is very expensive and time-consuming. Therefore, many approaches tried to relax this requirement and proposed methods that can be trained using weaker forms of supervision.

Earlier approaches for weakly supervised action segmentation apply discriminative clustering to detect actions using movie scripts (*Duchenne et al.*, 2009). However, these approaches ignored the action ordering information and only focused on distinguishing action segments from background. *Bojanowski et al.* (2014) extended these ideas to segment actions in videos using transcripts in the form of an ordered list of actions as supervision. This form of supervision has been very popular and many approaches have been proposed to train action segmentation models using transcripts as a supervisory signal. *Huang et al.* (2016) align the video frames and the transcripts using connectionist temporal classification. *Richard et al.* (2018b) combine a frame-wise loss function with the Viterbi algorithm to generate frame-wise target labels. *Chang et al.* (2019) apply a soft dynamic time warping algorithm and discriminate between positive and negative transcripts. In (*Li et al.*, 2019), an energy-based model is used such that the energy of all valid alignments that are consistent with the groundtruth transcripts is minimized. Other approaches generate pseudo groundtruth labels for

the training videos and iteratively refine them (*Kuehne et al.*, 2017; *Richard et al.*, 2017; *Ding and Xu*, 2018; *Kuehne et al.*, 2020). Starting with a uniform segmentation, *Kuehne et al.* (2017) trained a hidden Markov model (HMM) for each activity and a Gaussian mixture model (GMM) for the observation probabilities. The trained model is then used to infer a new alignment which is again used to retrain the model. The process is iteratively repeated until convergence. This work is extended in (*Richard et al.*, 2017; *Kuehne et al.*, 2020) by replacing the GMM with a GRU and further dividing the actions into smaller sub-actions to better model complex and long actions. In (*Ding and Xu*, 2018), soft labels are used around the boundaries between the action segments to reduce sensitivity to initialization, which increased the robustness of the training. While these approaches have been very successful, most of them suffer from a slow inference time as they iterate over all the training transcripts and select the one with the highest score. *Souri et al.* (2021) addressed this issue by predicting the transcript besides the frame-wise scores and further introduced a mutual consistency loss that enforces agreement between the predicted transcript and the frame-wise scores. Although these approaches rely on a cheap transcript supervision, their performance is much worse than fully supervised approaches.

Another line of research addresses the action segmentation task from set-level supervision. In contrast to transcripts, set supervision only provides the set of actions that occur in the videos without any information regarding the order or how many times each action occurs. Compared to transcripts, the space of possible labellings of a video given the groundtruth action set is very large. *Richard et al.* (2018a) proposed to reduce the search space of possible labellings by restricting the space of valid sequences of actions and their durations. *Li and Todorovic* (2020) combine a frame-wise classifier with a set-constrained Viterbi algorithm that only considers actions appearing in the groundtruth action set to generate frame-wise pseudo labels. This work has been extended in (*Li and Todorovic*, 2021) by considering the most salient frames for each action in the groundtruth action set as an anchor point to generate the pseudo labels. In contrast to these approaches that generate pseudo labels, *Fayyaz and Gall* (2020) trained a model that directly predicts action labels for temporal regions in the input videos and trained it with a set loss. While these approaches demonstrated that it is possible to train a segmentation model using only action sets, the proposed weaker level of supervision comes with the cost of inferior performance.

2.3 Timestamp Supervision for Recognizing Activities

Timestamp supervision has not yet received much attention from the action recognition community. Initial attempts were inspired by the success of point supervision for semantic segmentation (*Bearman et al.*, 2016). *Mettes et al.* (2016) apply multiple instance learning for spatio-temporal action localization using points annotation on a sparse subset of frames. *Chéron et al.* (2018) use discriminative clustering to integrate different types of supervision for the spatio-temporal action localization task. Recently, *Moltisanti et al.* (2019) proposed a sampling distribution based on a plateau function centered around temporal timestamps annotations to train a fine-grained action classifier. This approach relies on the classifier response to sample frames around the annotated timestamps and uses them for training. The method was tested for classifying trimmed videos and also showed promising results for temporal action localization. *Ma et al.* (2020) extended the action localization setup by mining action frames and background frames for training.

2.4 Action Anticipation

The interest in predicting future actions started at early stage. However, instead of predicting the future, initial approaches tried to recognize an ongoing activity given only a partial observation (*Ryoo*, 2011; *Hoai and De la Torre*, 2014). This setup goes in the literature by the name early action detection. Recent approaches addressed this task by using recurrent neural networks with special loss functions to encourage early activity detection (*Ma et al.*, 2016; *Sadegh Aliakbarian et al.*, 2017), predicting future representations of the unobserved frames as an auxiliary task (*Gammulle et al.*, 2019b), or by learning representations that correlate past with the future using knowledge distillation losses (*Tran et al.*, 2019; *Fernando and Herath*, 2021).

In contrast to the early action detection task, action anticipation predicts future actions without even observing any parts of these actions. Based on the predicted time horizon, action anticipation approaches can be grouped into two categories: short-term anticipation approaches and long-term anticipation approaches. While short-term approaches predict a single action a few seconds into the future, long-term approaches predict a sequence of future actions with their durations up to several minutes into the future. In the following sections, we discuss state-of-the-art approaches in both categories.

2.4.1 Short-Term Anticipation

Short-term anticipation approaches rely on the most recently observed frames to predict what will happen next. Early approaches focused on encoding the observations in a representation that facilitates predicting the future action. *Lan et al.* (2014) predicted future actions from hierarchical representations of short clips or still images. They encoded the observed frames in a multi-granular hierarchical representation of movements, and trained different classifiers at each level in the hierarchy in a max-margin framework. *Koppula and Saxena* (2016) used a spatio-temporal graph representation to encode the observed activities and then predict object affordances, trajectories, and sub-activities. Instead of directly predicting the future action labels, several approaches were proposed to predict future visual representations and then a classifier is trained on top of the predicted representations to predict the future labels. *Vondrick et al.* (2016) trained a deep network to predict visual representations of future frames, and used a support vector machine (SVM) to predict actions one second into the future. This work has been extended in *Gao et al.* (2017c), where an encoder-decoder network was proposed to predict a sequence of future representations based on a history of previous frames. To supervise the generated sequences, a reinforcement learning module was used to give high rewards to sequences that correspond to early correct predictions. Similarly, *Zeng et al.* (2017) used inverse reinforcement learning for visual sequence forecasting. *Rodriguez et al.* (2018) encoded human motions using dynamic images (*Bilen et al.*, 2016) and then predict the future representations from partial observations of human actions. In (*Shi et al.*, 2018), a recurrent neural network with a mapping layer consisting of a linear combination of Gaussian radial basis function (RBF) kernels was introduced for anticipating actions. *Surís et al.* (2021) proposed to learn hierarchical representations for the anticipation task based on the hyperbolic geometry. Besides action labels, some approaches used multi-task learning to predict the future action and its starting time (*Mahmud et al.*, 2017; *Mehrasa et al.*, 2019), or the future action and its spatial location (*Liang et al.*, 2019; *Sun et al.*, 2019a).

There is a parallel line of research addressing the anticipation task in egocentric videos. In (*Damen et al.*, 2018) the EPIC-KITCHENS dataset was introduced with a dedicated action anticipation challenge. As a baseline, a temporal segment network (TSN) (*Wang et al.*, 2016) was trained to directly predict an action label one second into the future. *Furnari and Farinella* (2019) proposed an LSTM-based sequence-to-sequence model to predict the future action at different time steps before its start. *Miech et al.* (2019) modeled the transition probabilities between actions and combine it with a predictive model to anticipate future action labels. *Sener et al.* (2020) proposed a multi-scale temporal aggregation framework for learning representations. The learned representations were used for both short-term and long-term anticipation. *Zatsarynna et al.* (2021) introduced a temporal convolutional network with a multi-modal fusion mechanism to predict actions one second into the future. To exploit the characteristics of egocentric videos, *Liu et al.* (2020) utilized the trajectories of hands as a representation for anticipating the future action. Similarly, *Dessalene et al.* (2021) exploited hands and objects segmentation masks and used the time to contact between hands and objects as a representation for the anticipation task. Recent approaches proposed transformer-based models for the anticipation task (*Girdhar and Grauman*, 2021; *Tai et al.*, 2021). In contrast to these approaches, we address the anticipation task for a longer time horizon.

2.4.2 Long-Term Anticipation

The long-term action anticipation task has been proposed in our work (*Abu Farha et al.*, 2018), which will be discussed in Chapter 6. We proposed two models to predict a sequence of future activities with their durations: an RNN-based and a CNN-based models. Both models were trained in a two-step approach by first inferring the activities in the observed part, then anticipating the future activities with their corresponding duration given the inferred activities. Inspired by these models, several approaches extended our approach and further improved the results. *Ke et al.* (2019) proposed to use temporal convolutions with attention to anticipate future activities. Instead of recursively predicting future activities, the model is conditioned on a time variable and only generates a prediction for that point in time. In (*Gammulle et al.*, 2019a), a recurrent neural network with memory is used to anticipate future activities from the ground-truth action labels of the observed frames. *Sener et al.* (2020) proposed a multi-scale temporal aggregation framework for learning representations by relating recent observations to the long-range past. The learned representations are then used as input to an LSTM to generate the sequence of future activities. *Piergiovanni et al.* (2020) proposed an adversarial generative grammar model and applied it for both action anticipation and forecasting 3D human poses. *Morais et al.* (2020) re-annotated the Breakfast dataset (*Kuehne et al.*, 2014) with hierarchical labels and proposed a hierarchical neural network model that predicts future actions at multiple levels of granularity.

Despite the success of the previous approaches in predicting the future actions, they only predict a single possible sequence of future actions. In (*Abu Farha and Gall*, 2019b), we proposed to predict multiple possible sequences to account for the uncertainty in the future. This approach will be discussed in Chapter 7. *Zhao and Wildes* (2020) built on our approach and proposed a conditional generative adversarial network with a normalized distance regularizer that encourages realistic and diverse predictions.

2.5 Anticipation Beyond Activity Labels

Predicting the future has become one of the main research topics in computer vision. A popular research direction focuses on predicting future frames in videos at pixel level. One of the early attempts on full resolution inputs was introduced by *Ranzato et al.* (2014), where recurrent models from natural language processing were used to predict future frames in video sequences. Recurrent neural networks were also used in (*Srivastava et al.*, 2015). In (*Mathieu et al.*, 2016), a multi-scale fully convolutional neural network was trained with an adversarial loss to generate sharp predictions. *Liang et al.* (2017) introduced a dual motion generative adversarial network (GAN) that predicts both the future frames and the corresponding future optical flow. To improve the quality of the predicted frames, several approaches proposed to first predict future human poses then condition the predicted frames on the generated poses (*Villegas et al.*, 2017; *Walker et al.*, 2017). Recent approaches tried to leverage physical knowledge to improve the anticipation approaches (*de Bezenac et al.*, 2018; *Guen and Thome*, 2020). While the quality of the predictions has significantly increased, low level representations like pixels of frames cannot be used directly for realistic applications. Instead of predicting pixel values, many approaches have been proposed to predict future semantic image segmentations (*Luc et al.*, 2017; *Nabavi et al.*, 2018; *Bhattacharyya et al.*, 2019), or instance segmentations (*Luc et al.*, 2018; *Sun et al.*, 2019b). Recently, *Graber et al.* (2021) combined these two representations and introduced the panoptic segmentation forecasting task.

Anticipating other representations has also been popular, such as future human trajectories (*Kitani et al.*, 2012; *Alahi et al.*, 2016), future human poses (*Jain et al.*, 2016; *Martinez et al.*, 2017; *Ruiz et al.*, 2019), traffic accidents (*Suzuki et al.*, 2018), human intentions (*Wu et al.*, 2017; *Rasouli et al.*, 2019), failure prediction (*Hecker et al.*, 2018; *Epstein et al.*, 2020), anticipation of instrument usage in surgical videos (*Rivoir et al.*, 2020), or even full sentences describing future frames or steps in recipes (*Sener and Yao*, 2019; *Mahmud et al.*, 2021). In contrast to these approaches, we focus on predicting future action labels as a representation.

2.6 Datasets

2.6.1 Breakfast

The Breakfast dataset (*Kuehne et al.*, 2014) is a collection of around 77 hours of video. In total, it consists of 1712 video clips, each video shows one out of ten possible activities related to preparing breakfast, such as making coffee or frying eggs. The activities were carried out by 52 different subjects and captured in 18 different kitchens. These activities are composed of sequences of more fine-grained actions like *take bowl* and *pour cereals*. In total, there are 48 such actions including background. For our experiments, we consider these specific 48 actions. The average length of the videos is around 2.3 minutes, with a temporal resolution of 15 frames per second (fps). Figure 2.1 shows some frames from the Breakfast dataset.

2.6.2 50Salads

The 50Salads dataset (*Stein and McKenna*, 2013) consists of 50 videos of preparing mixed salads. These videos were carried out by 25 subjects, where each subject prepares 2 salads. The dataset

Figure 2.1: Examples of frames from the Breakfast dataset. Each row depicts frames sampled from the same video.

provides fine-granular annotations of 17 low level activities, and also a coarser one with three high level activities. The dataset comes also with accelerometer data and depth images. However, in our experiments, we consider only the annotated RGB frames with the fine-granular annotations. The video sequences consist of actions related to preparing salads, such as *peel cucumber* and *mix ingredients*. The average length of the videos is around 6.4 minutes, with a temporal resolution of 30 fps. Figure 2.2 shows some frames from the 50Salads dataset.

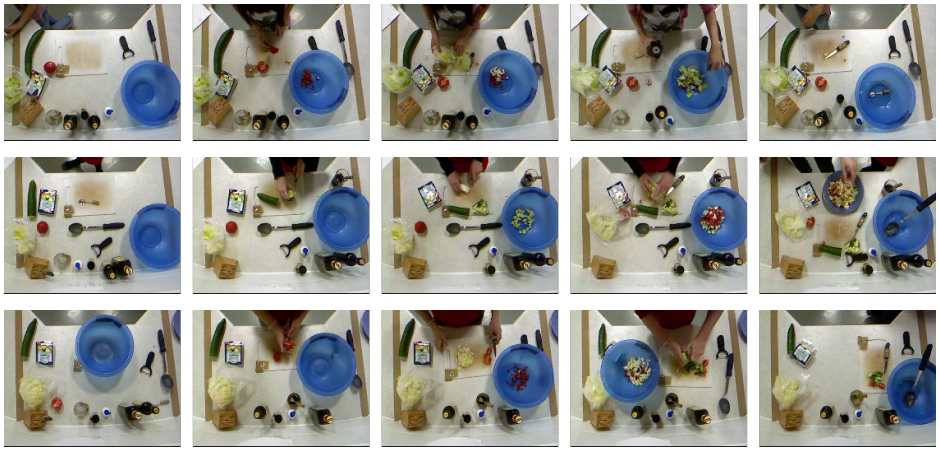


Figure 2.2: Examples of frames from the 50Salads dataset. Each row depicts frames sampled from the same video.

2.6.3 Georgia Tech Egocentric Activities (GTEA)

The GTEA dataset (*Fathi et al.*, 2011b) contains 28 videos corresponding to 7 different activities, such as preparing coffee or cheese sandwich, performed by 4 subjects. This dataset contains egocentric videos recorded by a camera that is mounted on the actor's head. The frames of the videos

are annotated with 11 action classes including background. On average, each video has 20 action instances. Figure 2.3 shows some frames from the GTEA dataset.

Figure 2.3: Examples of frames from the GTEA dataset. Each row depicts frames sampled from the same video.

2.6.4 Bristol Egocentric Object Interactions Dataset (BEOID)

The BEOID dataset (*Damen et al.*, 2014) contains 58 egocentric videos. The videos were recorded in six locations: kitchen, workspace, laser printer, corridor with a locked door, cardiac gym, and weight-lifting machine. Originally, the dataset has been used for task-relevant object discovery. Afterwards, the video frames were annotated with 34 action classes and have recently been used for evaluating weakly supervised action recognition approaches trained with timestamps annotations. Figure 2.4 shows some frames from the BEOID dataset.

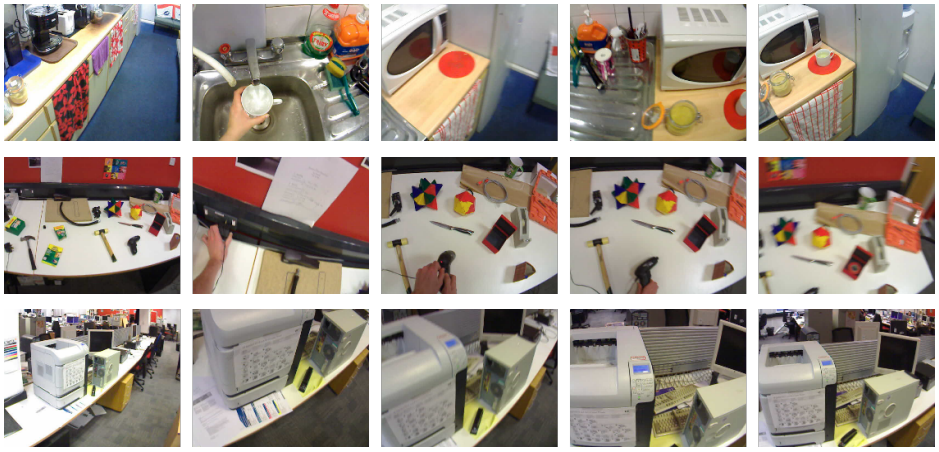


Figure 2.4: Examples of frames from the BEOID dataset. Each row depicts frames sampled from the same video.

CHAPTER 3

Preliminaries

In this chapter, we provide a brief overview of some concepts that are used throughout this thesis. We start with an introduction to neural networks, convolutional neural networks, and temporal modelling with recurrent neural networks and temporal convolutional networks. Then we describe how videos are represented and how to extract frame-wise features. Finally, we formally define the evaluation metrics for both the temporal action segmentation task and the long-term anticipation task.

Contents

3.1 Neural Networks

A neural network is a model that consists of interconnected layers, where each layer receives connections from the layer before. Each layer contains multiple processing units that are called neurons, which apply transformations to the output of the previous layer. Layers of a neural network can be grouped into three different groups: an input layer, hidden layers, and an output layer. The first layer is the input layer, and the last layer is the output layer. Layers between input and output layers are called hidden layers. In the standard form of the neural networks, the nodes and edges form a directed acyclic graph. This kind of neural networks is called a feed-forward neural network or multi-layer perceptron (MLP). Figure 3.1 shows the architecture of a feed-forward neural network with two hidden layers. Neural networks are used to approximate functions, and they are very popular in the field

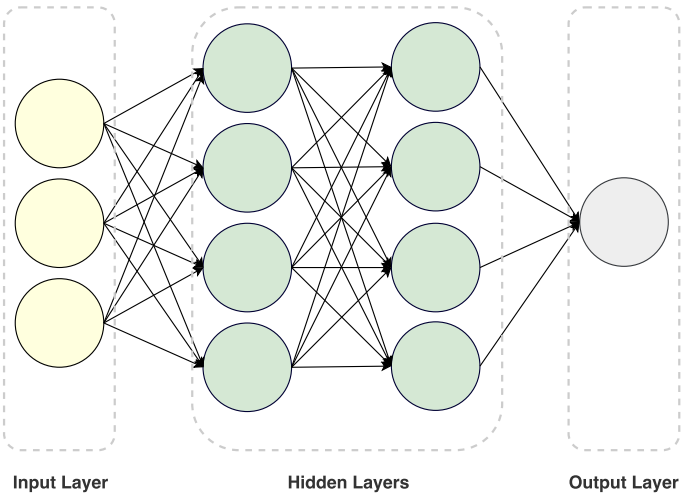


Figure 3.1: A feed-forward neural network with two hidden layers. Each layer consists of multiple neurons. Each neuron is connected to all neurons form the previous layer.

of pattern recognition, where it is infeasible to describe the solution as set of rules. In the following, we briefly describe the different components of neural networks.

3.1.1 Neuron

The neuron is the building block of neural networks. It can be viewed as a system that takes inputs, applies a transformation on them, and produces an output which is called activation. First it computes a weighted sum of the inputs, adds a bias, and then maps the final sum to an output value using its activation function, which is differentiable, usually non-linear, function. Feed-forward neural networks are constructed by stacking layers of neurons. Each neuron computes a weighted sum of the activations from the layer before and applies some non-linear transformation on the computed sum to produce an activation for the next layer as follows

$$o_j^{(l)} = b_j^{(l)} + \sum_{i=1}^{K} w_{ij}^{(l)}\, a_i^{(l-1)}, \tag{3.1}$$

$$a_j^{(l)} = f(o_j^{(l)}), \tag{3.2}$$

where $a_j^{(l)}$ is the activation of neuron j at layer l, $o_j^{(l)}$ is the weighted sum of activations from layer $l-1$, $w_{ij}^{(l)}$ is the weight from neuron i at layer $l-1$ to neuron j at layer l, and K is the total number of neurons at layer $l-1$. In (3.2), f is a non-linear activation function. It is crucial to have such non-linearity in intermediate layers. Otherwise, the transformation applied by the network would be equivalent to a single layer network. Figure 3.2 shows the architecture of a neuron with its inputs and output, and Figure 3.3 shows some of the most commonly used activation functions.

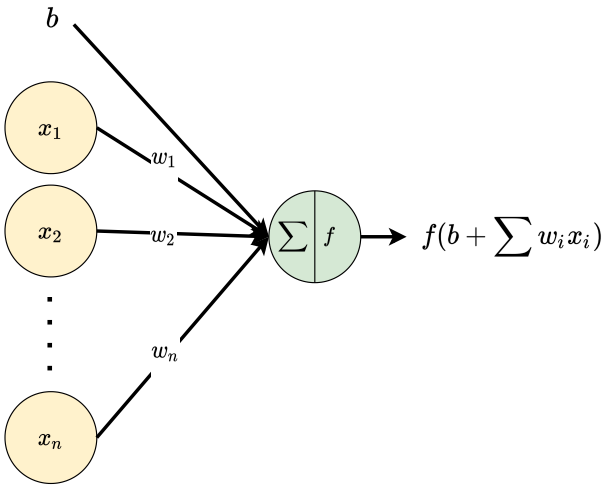


Figure 3.2: A neuron with n inputs $(x_1, \ldots, x_n)$, weights $(w_1, \ldots, w_n)$, a bias (b) and activation function f.

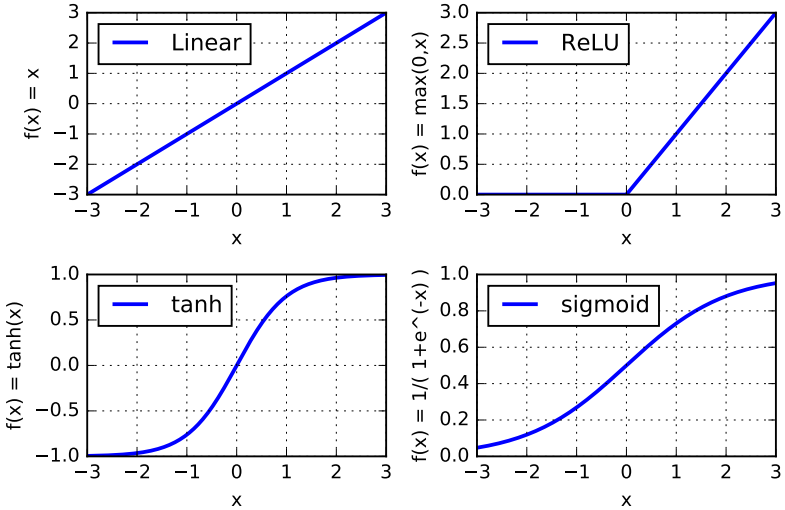


Figure 3.3: Different activation functions: Linear, Rectified Linear Unit (ReLU), tanh, and sigmoid.

3.1.2 Output Layer

The output layer is the last layer in the network that generates the final prediction for the corresponding input. The selection of the activation function for the output layer depends on the task. For regression tasks where the output is a real number, a linear activation function is used

$$f_{linear}(x) = x, \tag{3.3}$$

whereas for classification tasks, the activation function depends on the number of the classes. In the case of binary classification where the number of classes is only two, a sigmoid activation is used

$$f_{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \tag{3.4}$$

such that the output of the neuron is scaled to have a value between 0 and 1 to represent the probability of the corresponding class. Similarly, for multi-class classification problems, a softmax layer is used to ensure that the network output is a proper probability distribution over the different classes

$$f_{softmax}(o_i) = \frac{e^{o_i}}{\sum_j e^{o_j}}, \tag{3.5}$$

where o_i is the output of the $i\text{-}th$ neuron in the output layer. Note that softmax is only used if the target is a single label. For a multi-label setup, a sigmoid activation is used.

Furthermore, the activation function of the output layer can be used to enforce some constraints on the output values. For instance, an exponential activation can be used to enforce positive predictions, whereas a tanh activation restricts the output of the network to be between -1 and 1.

3.1.3 Objective Function

The objective function is a function that quantitatively measures the discrepancy between the neural network output and the desired one. Since it reflects the model performance, the objective function is also called loss function or error function. This function guides the neural network training to find a good set of parameters that minimize the error by following the negative direction of the objective function gradients. Therefore, the objective function has to be differentiable with respect to the network outputs. A commonly used objective function is the mean squared error (MSE)

$$\mathcal{L}_{mse}(\theta) = \frac{1}{N} \sum_{n=1}^{N} (y(x_n, \theta) - t_n)^2, \tag{3.6}$$

where the objective function $\mathcal{L}_{mse}$ is parameterized by the network parameters θ (the weights and biases of the network), N is the number of training examples, $y(x_n, \theta)$ is the output corresponding to the $n\text{-}th$ training example, and t_n is the target or desired value for that example. For multi-class classification tasks with C classes, the cross-entropy loss is usually used

$$\mathcal{L}_{ce}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} t_{n,c}\, log(y_c(x_n, \theta)), \tag{3.7}$$

where the target t_n is represented by a one-hot encoding of the target class and $t_{n,c}$ is its value for class c. Similarly, $y_c(x_n, \theta)$ is the predicted probability for class c. Since the target is a one-hot encoding vector, this loss can be written simply as

$$\mathcal{L}_{ce}(\theta) = -\frac{1}{N}\sum_{n=1}^{N} log(y_c(x_n, \theta)), \tag{3.8}$$

where $y_c(x_n, \theta)$ is the predicted probability for the target class c.

In general, any function that measures the discrepancy between the network output and some target value can be used as an objective function to train a neural network, given that it is differentiable with respect to the network parameters.

3.1.4 Network Training

In the previous sections, we described how neural networks stack layers of transformations to map the inputs to the desired outputs. The parameters of these transformations are learned from data by searching the parameters space for a set of parameters (weights and biases) that minimize the objective function. Gradient descent based algorithms are commonly used to train neural networks. Starting with an initial set of parameters, the learning algorithm iteratively updates these parameters by following the opposite direction of the gradients of the objective function with respect to the network parameters

$$\theta_{i+1} = \theta_i - \eta \nabla \mathcal{L}(\theta_i), \tag{3.9}$$

where θ_i are the network parameters at the $i\text{-}th$ iteration, $\mathcal{L}$ is the objective function, η is the learning rate, which is a small positive number that controls how much we move in the direction of the negative gradients. To compute the gradients with respect to the parameters at the intermediate layers, the backpropagation algorthim is used. The backpropagation algorithm recursively applies the chain rule such that the gradients with respect to the parameters at layer l are defined based on the gradients at layer $l + 1$.

The choice of the learning rate η is crucial for the convergence of the training. A large value of η can result in oscillations around the minimum value. On the contrary, if η is too small, the training would take too long to converge. Recent methods use an adaptive learning rate for different parameters based on the gradients moments such as the Adam optimizer (*Kingma and Ba*, 2015).

3.2 Convolutional Neural Networks (CNNs)

Traditional neural networks consist of fully connected layers, where each neuron receives connections from all neurons at the previous layer. This means that each layer is represented by a matrix multiplication of a weight matrix and an activation vector from the previous layer. However, convolutional networks are those networks that contain at least one layer of convolution. In contrast to fully connected layers, convolutional layers exhibits sparse connectivity, where each neuron is connected to a local region from the layer before. Moreover, neurons at the same layer share their parameters, which results in a significant reduction in the number of parameters compared to fully connected layers.

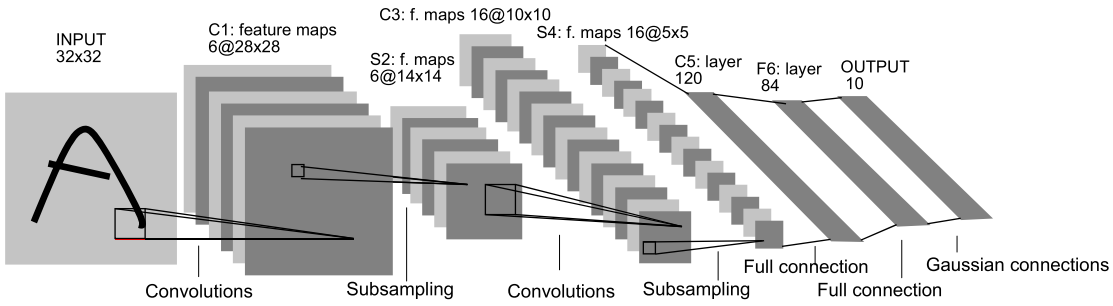


Figure 3.4: The network architecture of LeNet-5 (*LeCun et al.*, 1998).

Convolutional neural networks are widely applied on images, and show huge success in image classification and recognition. Each layer can be seen as a learned filter, or kernel, that extracts local features in the input image. As the parameters are shared between neurons operating at different regions, it is possible to learn multiple kernels with a relatively small number of parameters. The convolutional layer is then just a convolution operation between the input image I and the learned kernel W. A two-dimensional convolution is defined as follows

$$(I * W)(i, j) = \sum_{l,m} I(i - l, j - m)W(l, m). \tag{3.10}$$

Usually, convolutional networks consist of several convolutional layers followed by fully connected layers at the end. After the convolutional layers, a non-linearity is applied using some nonlinear activation like the ReLU. Sometimes a pooling layer is applied, where the activations of multiple neurons are reduced to a single one. Using such kind of layers, like max pooling, has the advantage of providing invariance to small translations.

There has been many successful convolutional neural network architectures. One of the popular 2D CNNs is the LeNet-5 (*LeCun et al.*, 1998) (see Figure 3.4), which is one of the first convolutional networks and has been used to recognize handwritten digits. AlexNet (*Krizhevsky et al.*, 2012) and VGG (*Simonyan and Zisserman*, 2015) have also been popular and applied to several tasks such as image classifications and object detection. Nowadays, the most commonly used architectures are based on ResNet (*He et al.*, 2016).

3.3 Recurrent Neural Networks (RNNs)

A recurrent neural network is a neural network with feedback loops that connect the outputs of some neurons back to their inputs. Therefore, the neurons of an RNN form a directed cyclic graph. In contrast to the traditional feed-forward networks, RNNs maintain a hidden state that summarizes the previous inputs, which makes them suitable for modelling sequential data. For an input sequence $x_{1:T} = (x_1, \ldots, x_T)$, the RNN updates its state h_t for step t based on the previous state h_{t-1} and the input x_t at time t. A vanilla RNN combines h_{t-1} and x_t linearly as follows

$$h_t = f(W_x x_t + W_h h_{t-1} + b), \tag{3.11}$$

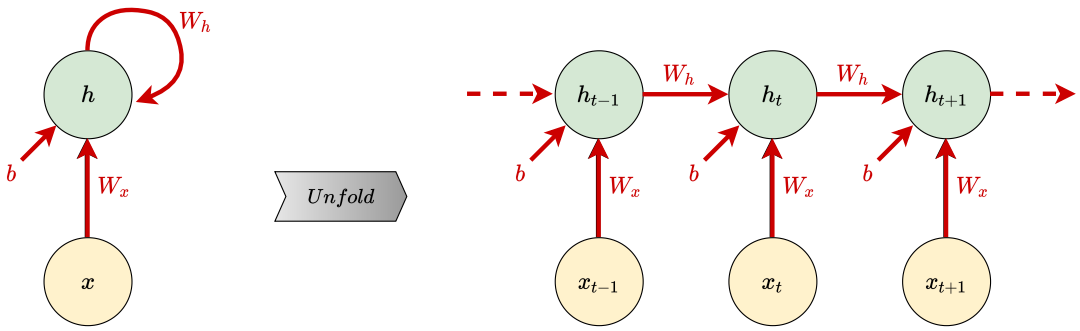


Figure 3.5: A vanilla recurrent neural network unfolded in time. The hidden state h_t at each step depends on the state from the previous time step h_{t-1} and the input x_t at the current step. The parameters W_x, W_h, and b are shared over time.

where W_x, W_h are weight matrices, b is a bias vector, and f is a non-linear activation function. The parameters W_x, W_h, and b are shared over time, which allows RNNs to handle input sequences with arbitrary length. To train recurrent neural networks, the back-propagation through time (BPTT) algorithm is used. First, the network is unfolded in time, where the same parameters are used in each step. Then, the back-propagation algorithm is used to update the network parameters. Figure 3.5 shows a vanilla RNN unfolded in time.

Despite the ability of RNNs to handle long sequences, training them with gradient descent based methods suffers from several problems such as vanishing or exploding gradients (*Bengio et al.*, 1994). To overcome these problems, several RNN variants have been developed by adopting multiplicative gates in the RNN update rule. The most popular RNN architectures are the long short-term memory (LSTM) (*Hochreiter and Schmidhuber*, 1997) and the gated recurrent unit (GRU) (*Cho et al.*, 2014).

3.3.1 Long Short-Term Memory (LSTM)

As mentioned previously, vanilla RNNs suffer from the vanishing gradients problem. This makes RNNs tend to forget quickly and restrict their ability to learn long temporal dependencies. To alleviate these problems, the LSTM adapts the recurrent architecture by introducing a memory cell that is updated at each step using multiplicative gates. The LSTM applies the following update rules at each time step

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \tag{3.12}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \tag{3.13}$$
$$c_t = f_t c_{t-1} + i_t \, tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \tag{3.14}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \tag{3.15}$$
$$h_t = o_t \, tanh(c_t), \tag{3.16}$$

where W_{**} are weight matrices, b_* are bias vectors, x_t is the input at time t, h_t is the hidden state at time t, c_t is the memory cell, σ is the sigmoid function, *i.e.* $\sigma(x) = \frac{1}{1+e^{-x}}$, and i_t, f_t, o_t are the input gate, forget gate, and output gate, respectively.

At each step, the forget gate determines what information should be removed form the memory cell and the input gate decides what should be added. Finally, the output is determined by multiplying the memory cell with the output gate.

3.3.2 Gated Recurrent Unit (GRU)

The GRU has been motivated by the LSTM. Nonetheless, it uses a simpler architecture with only two gates. The GRU applies the following update rules at each time step

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{3.17}$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \tag{3.18}$$
$$\tilde{h}_t = tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h), \tag{3.19}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \tag{3.20}$$

where $W_z,\ W_r,\ W_h,\ U_z,\ U_r,\ U_h,\ b_z,\ b_r,\ b_h$ are the parameters of the GRU, x_t is the input at time t, h_t is the hidden state at time t, σ is the sigmoid function, $\odot$ is the Hadamard product, $\tilde{h}_t$ is called the candidate state at time t, z_t is the update gate, and r_t is the reset gate.

Both the reset gate and the update gate control which information should flow to the next time step. The reset gate controls which information should be removed from the hidden state. Whereas the update gate controls how much information from the previous hidden state and the new candidate state should be transferred to the next time step.

3.4 Temporal Convolutional Networks (TCNs)

Temporal convolutional networks are a class of neural networks that are used to model sequential data. TCNs stack several layers of temporal convolutional layers. A temporal convolutional layer applies 1D convolution between the input sequence and a learned one dimensional kernel. For an input sequence $x_{1:T} = (x_1, \ldots, x_T)$ and a convolutional layer with kernel W of size k, the output of the convloutional layer h_t at time t is defined as follows

$$h_t = \sum_{i=0}^{k-1} x_{t-i} \cdot w_i, \tag{3.21}$$

where w_i is the $i\text{-}th$ parameter of the convolutional kernel W. The convolution defined in (3.21) is called causal since the output at time t only depends on inputs up to time t. It is also possible to define the convolution operation such that the output also depends on inputs from the future. In this case, the convolution is called acausal as illustrated in Figure 3.6.

Similar to traditional convolutional neural networks (CNNs), TCNs can also optionally have pooling layers. Other CNN-related concepts are also applicable to TCNs such as padding, strided convolution, and dilated convolutions. In this thesis, we consider the dilated convolution variant. A dilated convolution with dilation factor d is a standard convolution where the kernel is inflated by inserting $d-1$ zeros between the consecutive elements of the kernel. For example, a dilated convolution with kernel size 3 and dilation factor of 4 has a kernel of the form $(w_0, 0, 0, 0, w_1, 0, 0, 0, w_2)$.

In contrast to RNNs, the output of the TCNs at time t does not depend on the output at previous time steps and only depends on the input sequence. This allows for computing the output at all time steps in parallel, which makes TCNs much faster than RNNs, especially for long sequences.

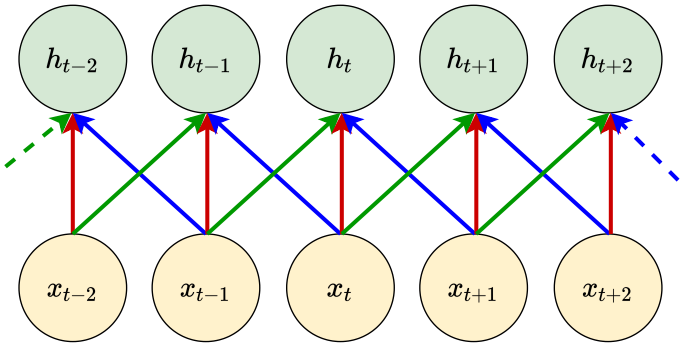


Figure 3.6: Acausal temporal convolution with kernel size 3. Links with the same color indicates sharing parameters.

3.5 Video Representation

A video is represented as a 4D tensor of dimension $3 \times H \times W \times T$, where T is the number of frames, $H \times W$ is the spatial dimension of each frame, and 3 is the three RGB channels. While deep learning approaches for action recognition takes the raw video frames as input, this is not feasible for the temporal action segmentation or the long-term anticipation tasks due to the limited resources. For these tasks, some videos might have more than ten thousands frames, which cannot be fitted in the limited memory of modern GPUs. Therefore, the video frames are pre-processed by a feature extraction algorithm and each frame is represented by an F-dimensional vector. *I.e.*, the video is now represented by an $F \times T$ tensor, which serves as input for the temporal action segmentation task. In this thesis, we work with two types of features representations: features based on improved dense trajectories and deep learned features using 3D CNNs. In the following, we provide more details about these features.

3.5.1 Improved Dense Trajectories (IDT)

Improved dense trajectories (*Wang et al.*, 2013; *Wang and Schmid*, 2013) have been very successful hand-crafted features for action recognition. The pipeline for extracting the dense trajectories is illustrated in Figure 3.7. First, dense feature points are sampled at different spatial scales. Then, these points are tracked through the video using optical flow. Finally, the trajectories are represented by different descriptors. In details, points are sampled on a grid for multiple spatial scales and tracked in each scale separately. To track the feature points, a dense optical flow field is first computed and smoothed by a median filter. For the improved dense trajectories, the flow is further corrected for the camera motion. Trajectories are then formed by concatenating tracked points of subsequent frames. To avoid drifting in tracking the feature points, the trajectories length is limited to L frames (for example $L = 15$). Furthermore, points sampled in homogeneous areas are discarded. Given the sequence of tracked points, each trajectory is encoded by a sequence of normalized displacement vectors. Furthermore, histograms of oriented gradients (HOG), histograms of optical flow (HOF), and motion boundary histograms (MBH) descriptors are computed in a 3D volume of size $N \times N \times L$ aligned around the trajectory. To capture structure information, the 3D volume is further divided into $n_\sigma \times n_\sigma \times n_\tau$ grid and the descriptors are computed for each cell separately as illustrated in Figure 3.7.

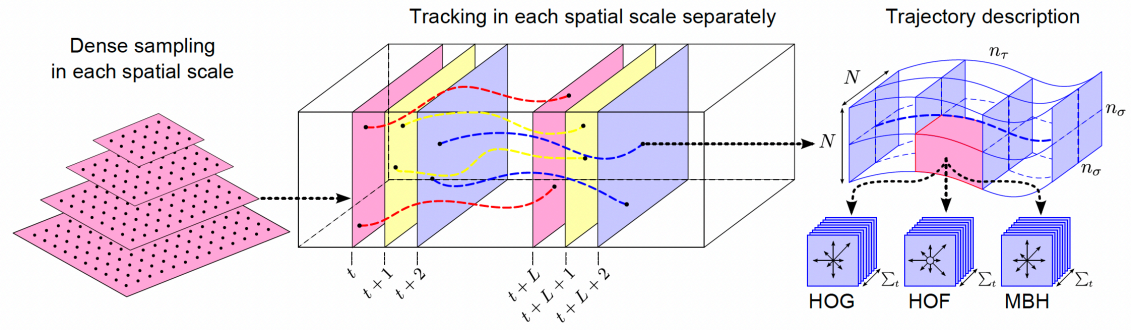


Figure 3.7: The pipeline for extracting dense trajectories from a video. Left: Dense sampling of feature points on a grid for different spatial scales. Middle: The points are tracked in each spatial scale for L frames based on a dense optical flow field. Right: The histograms of oriented gradients (HOG), histograms of optical flow (HOF), and motion boundary histograms (MBH) are computed along the trajectory in a $N \times N$ pixels neighborhood, which is divided into $n_\sigma \times n_\sigma \times n_\tau$ cells. The figure is taken from *Wang et al.* (2013).

In this thesis, we experimented with IDT-based features provided by *Kuehne et al.* (2016). First, the IDT features are computed as described in Figure 3.7. Then the dimensionality is reduced to 64 using principal component analysis (PCA). After that, the trajectories are encoded by the Fisher vectors (*Sánchez et al.*, 2013) based on a Gaussian mixture model (GMM) with 64 Gaussians. Fisher vectors is a technique to encode a patch of features by describing their deviation from a universal Gaussian mixture model. To compute these features, a GMM is first fitted to the features from all the training data. Then, given a patch of K features $x_{1:K}$, the average gradients of the log-likelihood of the features under the GMM is first computed with respect to the GMM parameters as follows

$$G_\lambda(x_{1:K}) = \frac{1}{K} \sum_{k=1}^{K} \nabla_\lambda \, log \, p_\lambda(x_k), \tag{3.22}$$

where $p_\lambda(x_k)$ is the likelihood of the feature x_k and λ represents the parameters of the GMM. The Fisher vector is then obtained by normalizing the gradients vector by the Fisher information matrix. More details on computing the Fisher vectors are given in (*Sánchez et al.*, 2013). In our setup, each frame in the input videos is represented by a feature vector based on the Fisher vectors encoding. To get the feature vector for a frame, the Fisher vectors are computed using all the trajectories that start in a window of 21 frames centered around that frame. Power- and ℓ_2-normalization are applied, and at the end, the dimensionality is again reduced to 64 with PCA.

3.5.2 Deep Learned Features

The progress of deep learning in modelling video data has been falling behind the success achieved for images. While initial attempts for using convolutional neural networks (CNNs) for video classification achieved much worse results compared to using IDT-based features (*Karpathy et al.*, 2014), it was until the introduction of the two-stream network (*Simonyan and Zisserman*, 2014) when a deep learning model could achieve a similar performance to the state-of-the-art hand-crafted features. Two-stream networks consist of two parallel CNNs. The first CNN is called the spatial or the appearance stream, which processes the RGB frames of the videos. Whereas the second stream is called

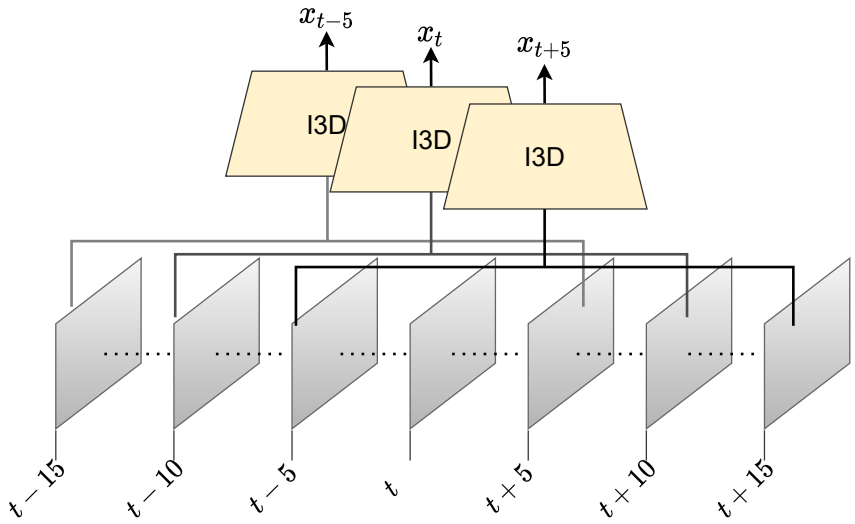


Figure 3.8: To compute the feature vector x_t for frame t with a window size of 21, all the frames in the interval $[t-10, t+10]$ are used as input to the I3D model.

the temporal stream, which takes a stack of optical flow frames. The output of these two streams is fused and then passed to a classifier. Several methods afterwards followed this two-stream architecture and continuously improved the results. In (*Carreira and Zisserman*, 2017), the two-stream architecture is extended to 3D CNNs where the learned convolutional kernels are three dimensional and operate both on the spatial and the temporal dimension. To alleviate the difficulty of designing and training 3D CNNs from scratch, *Carreira and Zisserman* (2017) proposed the I3D network by inflating a successful 2D CNN architecture into 3D and using its pre-trained weights on the ImageNet dataset (*Deng et al.*, 2009) for initializing the 3D CNN weights.

Pre-training deep networks on large-scale action classification datasets can be used to learn representations that are useful for other video understanding tasks. In this thesis, we use the I3D model pre-trained on the Kinetics-400 dataset (*Kay et al.*, 2017) to extract frame-wise features. To get a feature vector for a video frame, we forward a window of 21 frames centered around that frame to the pre-trained I3D model. We pass both the RGB frames and the optical flow frames and get the output of the penultimate layer from each stream. The final feature vector for that frame is then obtained by concatenating the output of both the appearance and temporal streams. Figure 3.8 shows the frames that are used to compute the feature vectors for each frame using a window of 21 frames.

3.6 Evaluation Metrics

3.6.1 Temporal Action Segmentation Metrics

Temporal action segmentation approaches use three different metrics: frame-wise accuracy, segmental edit score (Edit) and the segmental F1 score. In the following, we describe these metrics in detail.

Frame-wise Accuracy (Acc). Frame-wise accuracy measures the percentage of the correctly predicted frames, *i.e.*

$$Acc = \frac{\sum_{v=1}^{V} \sum_{t=1}^{T_v} \mathbb{1}_{a_{v,t}}[\hat{a}_{v,t}]}{\sum_{v=1}^{V} T_v} \cdot 100\%, \tag{3.23}$$

where $a_{v,t}$ is the ground-truth label at frame t in video v, $\hat{a}_{t,v}$ is the predicted label, T_v is the total number of frames in video v, and $\mathbb{1}_{a_{t,v}}[\hat{a}_{t,v}]$ is an indicator function which equals one if and only if the prediction is correct

$$\mathbb{1}_{a_{t,v}}[\hat{a}_{t,v}] = \begin{cases} 1 & if\ \hat{a}_{t,v} = a_{t,v} \\ 0 & if\ \hat{a}_{t,v} \neq a_{t,v} \end{cases}. \tag{3.24}$$

While the frame-wise accuracy (Acc) is the most commonly used metric for temporal action segmentation, long action classes have a higher impact than short action classes on this metric. Moreover, frame-wise metrics are not sensitive to segment-level errors such as over-segmentation.

Edit Score. the Edit score is a segmental metric that penalizes over-segmentation errors and out of order predictions. This metric is defined on a segment-level sequence of actions, which is referred to as the *transcript*. The transcript for a sequence of frame-wise labels can be obtained by squeezing consecutive frames with the same label. For instance, the sequence $\{a, a, a, b, b, c, c, c, c, c\}$ has a transcript of the form $\{a, b, c\}$. In contrast to the frame-wise accuracy, the edit score ignores the temporal shifts between the ground-truth and the predicted segments and only considers the sequential order of action segments. The edit score metric is defined based on the Levenshtein distance (*Levenshtein*, 1966), which measures the minimum number of character insertions, deletion or replacements that are required to change a sequence of characters into another target sequence. The Levenshtein distance between two sequences $\hat{A} = \{\hat{a}_1, \ldots, \hat{a}_n\}$ and $A = \{a_1, \ldots, a_m\}$ can be defined recursively as follows

$$L(\hat{A}, A) = \begin{cases} |\hat{A}| & if\ |A| = 0, \\ |A| & if\ |\hat{A}| = 0, \\ L(\{\hat{a}_2, \ldots, \hat{a}_n\}, \{a_2, \ldots, a_m\}) & if\ \hat{a}_1 = a_1, \\ 1 + min \begin{cases} L(\{\hat{a}_2, \ldots, \hat{a}_n\}, A) \\ L(\hat{A}, \{a_2, \ldots, a_m\}) \\ L(\{\hat{a}_2, \ldots, \hat{a}_n\}, \{a_2, \ldots, a_m\}) \end{cases} & otherwise, \end{cases} \tag{3.25}$$

where $L(.,.)$ is the Levenshtein distance, $|.|$ denotes the length of a sequence. The edit score is then computed based on the normalized Levenshtein distance

$$Edit = \left(1 - \frac{L(\hat{A}, A)}{max(|\hat{A}|, |A|)}\right) \cdot 100\%. \tag{3.26}$$

F1 Score. Similar to the edit score, the F1 score is also a segmental metric. However, unlike the edit score, the F1 score considers the temporal overlap between the ground-truth and the predicted segments. This metric is associated with a threshold k that is used to group the predicted segments into true positives (TP) and false positives (FP) based on the intersection over union (IoU) ratio,

$$IoU = \frac{predicted\ segment \cap ground\text{-}truth\ segment}{predicted\ segment \cup ground\text{-}truth\ segment}. \tag{3.27}$$

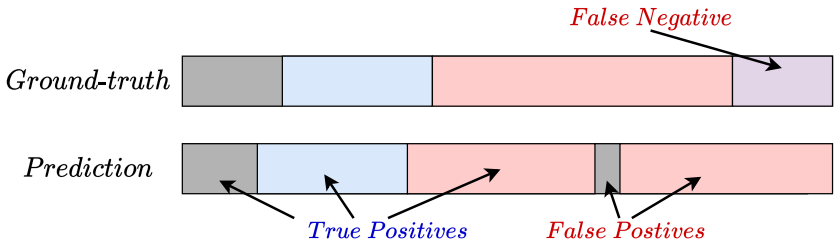


Figure 3.9: Examples of true positive, false positive and false negative segments for an intersection over union (IoU) ratio k of 25%. Note that both of the red segments have an IoU ratio higher than 25% but only the first one is considered as a true positive since it has a higher IoU ratio.

If a predicted segment has the same label as the ground-truth segment and the IoU ratio is higher than k, then it is considered a true positive. Otherwise, the segment is considered a false positive. If multiple predicted segments have an IoU ratio higher than k, then only the segment with the maximum ratio is considered a true positive and the others are labeled as false positives. After considering all the predicted segments, any ground-truth segment that is not associated with any predicted segment is considered a false negative (FN). Figure 3.9 illustrates examples of the different segment types for an IoU ratio of 25%. To compute the F1 score, we first calculate the precision and the recall as follows

$$Precision = \frac{TP}{TP + FP}, \tag{3.28}$$

$$Recall = \frac{TP}{TP + FN}, \tag{3.29}$$

then, the F1 score at overlapping threshold k is defined as

$$F1@k = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \cdot 100\%. \tag{3.30}$$

Since the F1 score is based on the IoU ratio, it does not depend on the duration of the action segments and treats both short and long segments equally. Furthermore, small temporal shifts between the predicted and the ground-truth segments are not penalized.

3.6.2 Long-Term Anticipation Metrics

The standard metric for the long-term action anticipation is the **mean over classes (MoC)**. This metric is defined by the average of the class-wise accuracy. *I.e.* the accuracy of each class is computed independently by calculating the percentage of the correctly predicted frames for that class. Then, the MoC is computed by taking the average of these accuracies over all classes as follows

$$MoC = \frac{1}{C} \sum_{c} Acc_c, \tag{3.31}$$

$$Acc_c = \frac{\sum_{v=1}^{V} \sum_{t=1}^{T_v} \mathbb{1}_{a_{v,t}}[\hat{a}_{v,t}] \cdot \mathbb{1}_c[a_{v,t}]}{\sum_{v=1}^{V} \sum_{t=1}^{T_v} \mathbb{1}_c[a_{v,t}]} \cdot 100\%, \tag{3.32}$$

where C is the total number of action classes, Acc_c is the frame-wise accuracy for class c, $a_{v,t}$ is the ground-truth label at frame t in video v, $\hat{a}_{v,t}$ is the predicted label, and T_v is the total number of

frames in video v. Averaging the accuracies over the action classes ensures that both short and long action segments contribute equally to the metric.

CHAPTER 4

Multi-Stage Temporal Convolutional Networks for Action Segmentation

In this chapter, we propose two multi-stage architectures for the temporal action segmentation task. In both architectures, the first stage generates an initial prediction that is refined by the next ones. To capture long-range dependencies, we stack several layers of dilated temporal convolutions in each stage covering a large receptive field with a few parameters. While the first architecture has a receptive field that exponentially increases with the number of layers, the second architecture combines both large and small receptive fields at each layer. We further propose a smoothing loss that penalizes over-segmentation errors and improves the quality of the predictions. Extensive evaluation shows the effectiveness of the proposed models in capturing long-range dependencies and recognizing action segments. Our models achieve state-of-the-art results on three datasets when the work has been published: 50Salads, Georgia Tech Egocentric Activities (GTEA), and the Breakfast dataset.

This chapter is based on (*Abu Farha and Gall*, 2019a; *Li et al.*, 2020). Yazan Abu Farha proposed the multi-stage architecture (MS-TCN), the smoothing loss, and wrote the papers. Shijie Li proposed the extension of MS-TCN, which we call MS-TCN++. Both authors contributed to the experiments.

Contents

4.1 Introduction

Action recognition from video has been an active research area in computer vision in the past few years. Most of the efforts, however, have been focused on classifying short trimmed videos (*Simonyan and Zisserman*, 2014; *Feichtenhofer et al.*, 2016; *Carreira and Zisserman*, 2017; *Feichtenhofer et al.*, 2019). Despite the success of these approaches on trimmed videos with a single activity, their performance is limited on long videos containing many action segments. Since for many applications, like surveillance and robotics, it is crucial to temporally segment activities in long untrimmed videos, approaches for temporal action segmentation have received more attention. Early attempts for temporal action segmentation tried to extend the success on trimmed videos by combining these models with sliding windows (*Rohrbach et al.*, 2012; *Karaman et al.*, 2014; *Oneata et al.*, 2014). These approaches use temporal windows of different scales to detect and classify action segments. However, such approaches are expensive and do not scale for long videos. Other approaches apply a coarse temporal modeling using Markov models on top of frame-wise classifiers (*Kuehne et al.*, 2016; *Lea et al.*, 2016; *Richard et al.*, 2017). While these approaches achieved good results, they are very slow as they require solving a maximization problem over very long sequences.

With the success of temporal convolutional networks (TCNs) as a powerful temporal model for speech synthesis, many researchers adapt TCN-based models for the temporal action segmentation task (*Lea et al.*, 2017; *Lei and Todorovic*, 2018; *Ding and Xu*, 2018). These models were more capable in capturing long range dependencies between the video frames by relying on a large receptive field. However, these models are limited for a very low temporal resolution of a few frames per second. Furthermore, since these approaches rely on temporal pooling layers to increase the receptive field, many of the fine-grained information that is necessary for recognition is lost.

To overcome the limitations of the previous approaches, we propose a new model that also uses temporal convolutions. In contrast to previous approaches, the proposed model operates on the full temporal resolution of the videos and thus achieves better results. Our model consists of multiple stages where each stage outputs an initial prediction that is refined by the next one. We call the new architecture Multi-Stage Temporal Convolutional Network (MS-TCN). In each stage, we apply a series of dilated 1D convolutions, which enables the model to have a large temporal receptive field with less parameters. Figure 4.1 shows an overview of the proposed multi-stage model. Furthermore, we employ a smoothing loss during training which penalizes over-segmentation errors in the predictions.

While the proposed MS-TCN already achieves good performance, some of the design choices are sub-optimal. First, although the receptive field is very large for higher layers in MS-TCN, lower layers suffer from a small receptive field. Second, the first stage generates an initial prediction, which is refined by the remaining stages. Despite the differences between these two tasks, all stages share the same architecture. To address the first limitation, we propose a dual dilated layer (DDL) which combines both large and small receptive fields at each layer. For the second, we divide the whole architecture into two parts: The first part is the first stage, which is the prediction generation stage, and the second part consists of prediction refinement stages. Then we customize the architecture of each part separately and do not force all stages to have the same architecture as in MS-TCN.

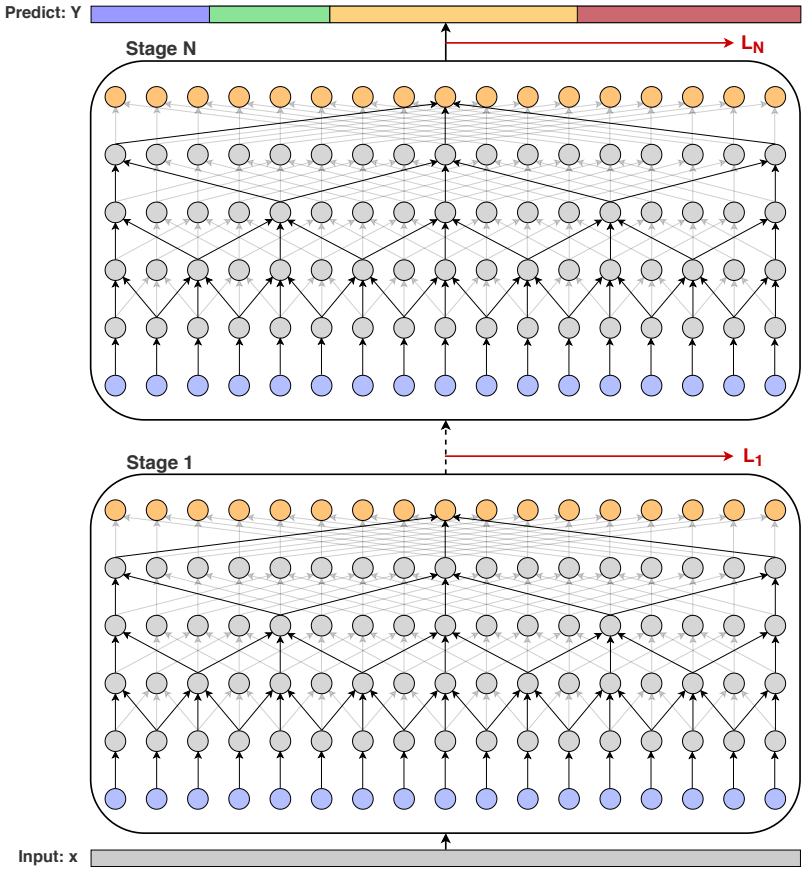


Figure 4.1: Overview of the multi-stage temporal convolutional network. Each stage generates an initial prediction that is refined by the next stage. At each stage, several dilated 1D convolutions are applied on the activations of the previous layer. A loss layer is added after each stage.

By incorporating these design choices on MS-TCN, we propose an improved version of the model, which we call MS-TCN++. Furthermore, we show that the parameters of the refinement stages in MS-TCN++ can be shared without compromising the accuracy. This model achieves superior performance compared to MS-TCN with much less parameters. Our contribution is thus five folded:

- We propose a multi-stage temporal convolutional network (MS-TCN) for the action segmentation task that operates on the full temporal resolution.
- We introduce a smoothing loss to enhance the quality of the predictions and reduce over-segmentation errors.
- We propose a dual dilated layer that combines large and small receptive fields.
- We optimize the architecture design of MS-TCN by decoupling the prediction phase and the refinement phase. We call the new model MS-TCN++, which achieves superior results compared to MS-TCN.
- We further show that sharing the parameters between the refinement stages in MS-TCN++ results in a more compact model without compromising performance.

Extensive evaluation shows the effectiveness of our models in capturing long range dependencies between action classes and producing high quality predictions. Our approach achieves state-of-the-art results on three challenging benchmarks for action segmentation: 50Salads (*Stein and McKenna*, 2013), Georgia Tech Egocentric Activities (GTEA) (*Fathi et al.*, 2011b), and the Breakfast dataset (*Kuehne et al.*, 2014). Moreover, the proposed models are view-agnostic and work well on all the three datasets, which depict videos with third person view, top view and egocentric videos.

4.2 Temporal Action Segmentation

We introduce a multi-stage temporal convolutional network (MS-TCN) for the temporal action segmentation task. Then we introduce a new layer and address the limitations of MS-TCN to propose an improved model called MS-TCN++. Given the frames of a video $x_{1:T} = (x_1, \ldots, x_T)$, our goal is to infer the class label for each frame $a_{1:T} = (a_1, \ldots, a_T)$, where T is the video length. First, we describe the single-stage approach in Section 4.2.1, then we discuss the multi-stage model in Section 4.2.2. Section 4.2.3 introduces the dual dilated layer. In Section 4.2.4, we analyze the drawbacks of MS-TCN and introduce the improved model MS-TCN++. Finally, we describe the proposed loss function in Section 4.2.5.

4.2.1 Single-Stage Temporal Convolutional Network (SS-TCN)

Our single stage model consists of only temporal convolutional layers. We do not use pooling layers, which reduce the temporal resolution, or fully connected layers, which force the model to operate on inputs of fixed size and massively increase the number of parameters. We call this model a single-stage temporal convolutional network (SS-TCN). The first layer of a single-stage TCN is a 1×1 convolutional layer, that adjusts the dimension of the input features to match the number of feature maps in the network. Then, this layer is followed by several layers of dilated 1D convolution. Inspired by the wavenet (*Van Den Oord et al.*, 2016) architecture, we use a dilation factor that is doubled at each layer, *i.e.* 1, 2, 4, ..., 512. All these layers have the same number of convolutional filters. However, instead of the causal convolution that is used in wavenet, we use acausal convolutions with kernel size 3. Each layer applies a dilated convolution with ReLU activation to the output of the previous layer. We further use skip connections to facilitate gradients flow. The set of operations at each layer can be formally described as follows

$$\hat{H}_l = ReLU(W_d * H_{l-1} + b_d), \tag{4.1}$$

$$H_l = H_{l-1} + W * \hat{H}_l + b, \tag{4.2}$$

where H_l is the output of layer l, $*$ denotes the convolution operator, $W_d \in \mathbb{R}^{3\times D\times D}$ are the weights of the dilated convolution filters with kernel size 3 and D is the number of convolutional filters, $W \in \mathbb{R}^{1\times D\times D}$ are the weights of a 1×1 convolution, and $b_d, b \in \mathbb{R}^D$ are bias vectors. These operations are illustrated in Figure 4.2. Using dilated convolution increases the receptive field without the need to increase the number of parameters by increasing the number of layers or the kernel size. Since the receptive field grows exponentially with the number of layers, we can achieve a very large receptive field with a few layers, which helps in preventing the model from over-fitting the training data. The receptive field at each layer is determined by

$$ReceptiveField(l) = 2^{l+1} - 1, \tag{4.3}$$

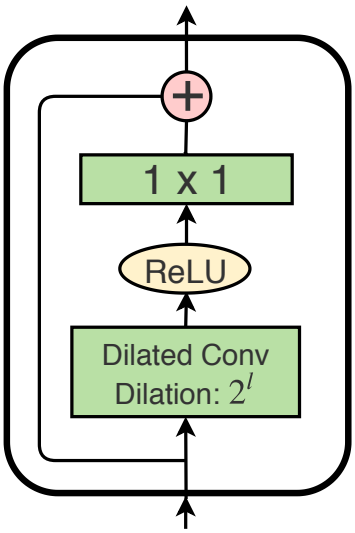


Figure 4.2: Overview of the dilated residual layer. At each layer l, the dilated residual layer uses a convolution with dilated factor 2^l.

where $l \in [1, L]$ is the layer number. Note that this formula is only valid for a kernel of size 3. To get the probabilities for the output class, we apply a 1×1 convolution over the output of the last dilated convolution layer followed by a softmax activation, *i.e.*

$$Y_t = Softmax(Wh_{L,t} + b), \tag{4.4}$$

where Y_t contains the class probabilities at time t, $h_{L,t}$ is the output of the last dilated convolution layer at time t, $W \in \mathbb{R}^{C \times D}$ and $b \in \mathbb{R}^C$ are the weights and bias for the 1×1 convolution layer. C is the number of classes and D is the number of convolutional filters.

4.2.2 Multi-Stage Temporal Convolutional Network (MS-TCN)

Stacking several predictors sequentially has shown significant improvements in many tasks like human pose estimation (*Wei et al.*, 2016; *Newell et al.*, 2016; *Dantone et al.*, 2014). The idea of these stacked or multi-stage architectures is composing several models sequentially such that each model operates directly on the output of the previous one. The effect of such composition is an incremental refinement of the predictions from the previous stages.

Motivated by the success of such architectures, we introduce a multi-stage temporal convolutional network (MS-TCN) for the temporal action segmentation task. In this multi-stage model, each stage takes an initial prediction from the previous stage and refines it. The input of the first stage are the frame-wise features of the video as follows

$$Y^0 = x_{1:T}, \tag{4.5}$$
$$Y^s = \mathcal{F}(Y^{s-1}), \tag{4.6}$$

where Y^s is the output at stage s and $\mathcal{F}$ is the single-stage TCN discussed in Section 4.2.1. Using such a multi-stage architecture helps in providing more context to predict the class label at each frame. Furthermore, since the output of each stage is an initial prediction, the network is able to

capture dependencies between action classes and learn plausible action sequences, which helps in reducing the over-segmentation errors.

Note that the input to the next stage is just the frame-wise probabilities without any additional features. We will show in the experiments how adding features to the input of the next stage affects the quality of the predictions.

4.2.3 Dual Dilated Layer

In the dilated convolution layers in MS-TCN, the dilation factor increases as we increase the number of layers. While this results in a large receptive field for higher layers, lower layers still suffer from very low receptive fields. Furthermore, higher layers in MS-TCN apply convolutions over very distant time steps due to the large dilation factor. To overcome this problem, we propose a dual dilated layer (DDL). Instead of having one dilated convolution, the DDL combines two convolutions with different dilation factor. The first convolution has a low dilation factor in lower layers and exponentially increases as we increases the number of layers. Whereas for the second convolution, we start with a large dilation factor in lower layers and exponentially decrease it with increasing the number of layers. The set of operations at each layer can be formally described as follows

$$\hat{H}_{l,d_1} = W_{d_1} * H_{l-1} + b_{d_1}, \tag{4.7}$$

$$\hat{H}_{l,d_2} = W_{d_2} * H_{l-1} + b_{d_2}, \tag{4.8}$$

$$\hat{H}_{l} = ReLU([\hat{H}_{l,d_1}, \hat{H}_{l,d_2}]), \tag{4.9}$$

$$H_l = H_{l-1} + W * \hat{H}_l + b, \tag{4.10}$$

where $W_{d_1}, W_{d_2} \in \mathbb{R}^{3\times D\times D}$ are the weights of dilated convolutions with dilation factor 2^l and 2^{L-l} respectively, $W \in \mathbb{R}^{1\times 2D\times D}$ are the weights of a 1×1 convolution, and $b_{d_1}, b_{d_2}, b \in \mathbb{R}^D$ are bias vectors. In (4.9), $\hat{H}_{l,d_1}$ and $\hat{H}_{l,d_2}$ are concatenated. An overview of the dual dilated layer is illustrated in Figure 4.3.

While the dual dilated layer combines local and global context from the input sequence, there are other techniques in the literature for fusing multi-scale features like feature pyramid networks (FPN) (*Lin et al.*, 2017). While applying FPN for temporal action segmentation has been successful (*Ding and Xu*, 2018), these approaches still suffer from a very limited receptive field. Moreover, the multi-scale features in FPN are obtained by applying pooling operations which results in a loss of the fine-grained information that is necessary for temporal segmentation. On the contrary, DDL combines multi-scale features and yet preserves the temporal resolution of the input sequence.

4.2.4 MS-TCN++

In this section, we introduce MS-TCN++, which utilizes the proposed dual dilated layer to improve MS-TCN. Similar to MS-TCN, the first stage in MS-TCN++ is responsible for generating the initial prediction, whereas the remaining stages incrementally refine this prediction. For the prediction generation stage, we adapt an SS-TCN with dual dilated layers (Figure 4.3) replacing the simple dilated residual layers (Figure 4.2) that are originally used in SS-TCN. Using the DDL enables the prediction generation stage to capture both local and global features in all layers, which results in better predictions. As refinement is easier than prediction generation, we adapt the SS-TCN architecture

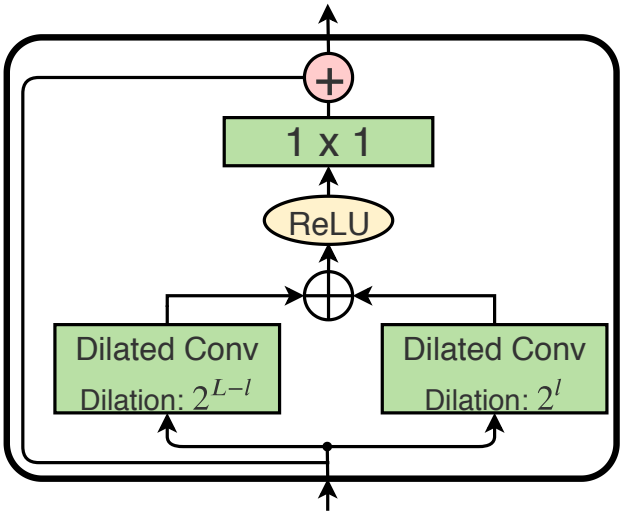


Figure 4.3: Overview of the dual dilated layer (DDL). At each layer l, DDL uses two convolutions with dilated factor 2^l and 2^{L-l}, respectively, where L is the number of layers in the network.

with dilated residual layers for the refinement stages. In our experiments, we show that using DDL only for the first stage performs best. Figure 4.4 shows an overview of the proposed MS-TCN++.

While adding more stages incrementally refines the predictions, it also drastically increases the number of parameters. Nevertheless, as the refinement stages are sharing the same role, their parameters can be shared to get a more compact model. In the experiments, we show that sharing the parameters between the refinement stages significantly reduces the number of parameters with only a slight degradation in accuracy.

4.2.5 Loss Function

As a loss function, we use a combination of a classification loss and a smoothing loss. For the classification loss, we use a cross entropy loss

$$\mathcal{L}_{cls} = \frac{1}{T}\sum_{t} -log(y_{t,a}), \tag{4.11}$$

where $y_{t,a}$ is the predicted probability for the ground truth label a at time t.

While the cross entropy loss already performs well, we found that the predictions for some of the videos contain a few over-segmentation errors. To further improve the quality of the predictions, we use an additional smoothing loss to reduce such over-segmentation errors. For this loss, we use a truncated mean squared error over the frame-wise log-probabilities

$$\mathcal{L}_{T-MSE} = \frac{1}{TC}\sum_{t,a} \tilde{\Delta}^2_{t,a}, \tag{4.12}$$

$$\tilde{\Delta}_{t,a} = \begin{cases} \Delta_{t,a} & : \Delta_{t,a} \leqslant \tau \\ \tau & : otherwise \end{cases}, \tag{4.13}$$

$$\Delta_{t,a} = \left|log\ y_{t,a} - log\ y_{t-1,a}\right|, \tag{4.14}$$

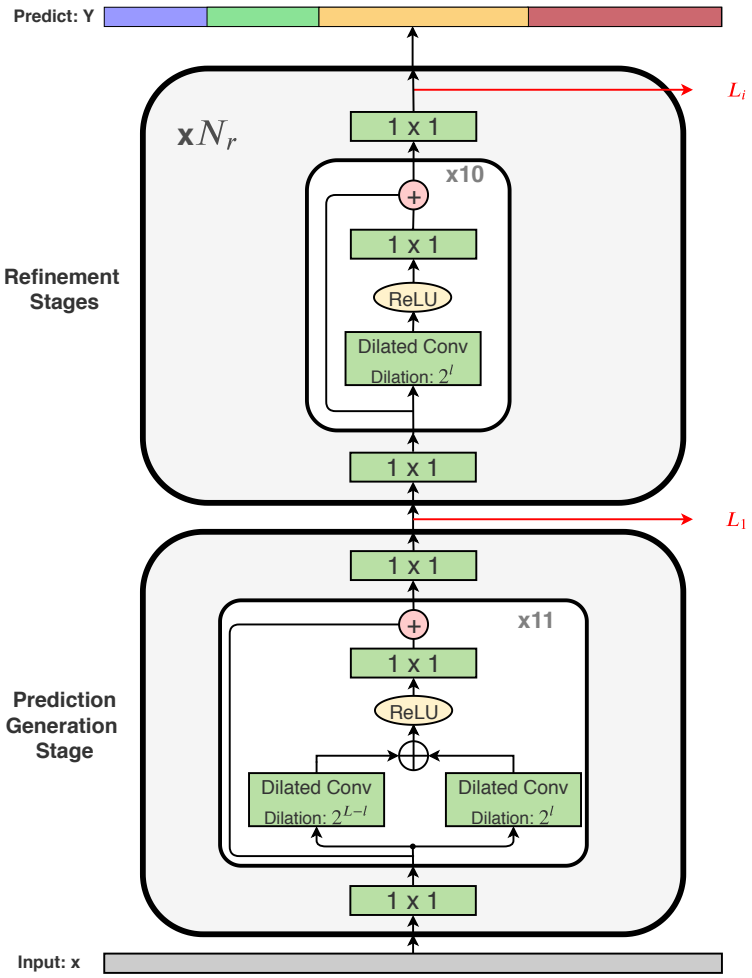


Figure 4.4: Overview of MS-TCN++. The first stage adapts an SS-TCN model with dual dilated layers. This stage generates an initial prediction that is refined incrementally by a set of N_r refinement stages. For the refinement stages, an SS-TCN with dilated residual layers is used. A loss layer is added after each stage.

where T is the video length, C is the number of classes, $y_{t,a}$ is the probability of class a at time t, and τ is a hyper-parameter.

Note that the gradients are only computed with respect to $y_{t,a}$, whereas $y_{t-1,a}$ is not considered as a function of the model's parameters. This loss is similar to the Kullback-Leibler (KL) divergence loss where

$$\mathcal{L}_{KL} = \frac{1}{T}\sum_{t,a} y_{t-1,a}(log\ y_{t-1,a} - log\ y_{t,a}). \tag{4.15}$$

However, we found that the truncated mean squared error ($\mathcal{L}_{T-MSE}$) (4.12) reduces the over-segmentation errors more. We will compare the KL loss and the proposed loss in the experiments.

The final loss function for a single stage is a combination of the above mentioned losses

$$\mathcal{L}_s = \mathcal{L}_{cls} + \lambda\mathcal{L}_{T-MSE}, \tag{4.16}$$

where λ is a model hyper-parameter to determine the contribution of the different losses. Finally to train the complete model, we minimize the sum of the losses over all stages

$$\mathcal{L} = \sum_{s} \mathcal{L}_s. \tag{4.17}$$

4.3 Experiments

Datasets. We evaluate the proposed models on three challenging datasets: 50Salads (*Stein and McKenna*, 2013), Georgia Tech Egocentric Activities (GTEA) (*Fathi et al.*, 2011b), and the Breakfast dataset (*Kuehne et al.*, 2014). Details about the datasets are available in Chapter 2.

For all datasets, we extract I3D (*Carreira and Zisserman*, 2017) features for the video frames and use these features as input to our model. For the GTEA and Breakfast datasets we use the temporal video resolution at 15 fps, while for 50Salads we downsampled the features from 30 fps to 15 fps to be consistent with the other datasets.

Evaluation Metrics. For evaluation, we report the frame-wise accuracy (Acc), segmental edit score (Edit), and the segmental F1 score at overlapping thresholds 10%, 25% and 50%, denoted by $F1@\{10, 25, 50\}$. The overlapping threshold is determined based on the intersection over union (IoU) ratio. While the frame-wise accuracy is the most commonly used metric for action segmentation, long action classes have a higher impact than short action classes on this metric and over-segmentation errors have a very low impact. For that reason, we use the segmental F1 score as a measure of the quality of the prediction as proposed by (*Lea et al.*, 2017). Formal definitions for the evaluation metrics are discussed in Chapter 3.

Implementation Details. For both MS-TCN and MS-TCN++, we use a multi-stage architecture with four stages. While all stages are the same for MS-TCN, the stages in MS-TCN++ consist of one prediction generation stage and three refinement stages. Each stage in MS-TCN and the refinement stages in MS-TCN++ contain ten dilated convolution layers. For the prediction generation stage in MS-TCN++, we use eleven layers. Dropout is used after each layer with probability 0.5. We set the number of filters to 64 in all layers of the model and the filter size is 3. For the loss function, we set $\tau = 4$ and $\lambda = 0.15$. In all experiments, we use Adam optimizer with a learning rate of 0.0005.

4.3.1 Effect of the Number of Stages

We start our evaluation by showing the effect of using a multi-stage architecture (MS-TCN). Table 4.1 shows the results of a single-stage model compared to multi-stage models with different number of stages. As shown in the table, all of these models achieve a comparable frame-wise accuracy. Nevertheless, the quality of the predictions is very different. Looking at the segmental edit distance and F1 scores of these models, we can see that the single-stage model produces a lot of over-segmentation errors, as indicated by the low F1 score. On the other hand, using a multi-stage architecture reduces these errors and increases the F1 score. This effect is clearly visible when we use two or three stages, which gives a huge boost to the accuracy. Adding the fourth stage still improves the results but not as significant as the previous stages. However, by adding the fifth stage, we can see that the performance starts to degrade. This might be an over-fitting problem as a result of increasing the number of parameters. The effect of the multi-stage architecture can also be seen in the qualitative results

	F1@{10,25,50}			Edit	Acc
SS-TCN	27.0	25.3	21.5	20.5	78.2
MS-TCN (2 stages)	55.5	52.9	47.3	47.9	79.8
MS-TCN (3 stages)	71.5	68.6	61.1	64.0	78.6
MS-TCN (4 stages)	76.3	**74.0**	**64.5**	67.9	**80.7**
MS-TCN (5 stages)	**76.4**	73.4	63.6	**69.2**	79.5

Table 4.1: Effect of the number of stages on the 50Salads dataset.

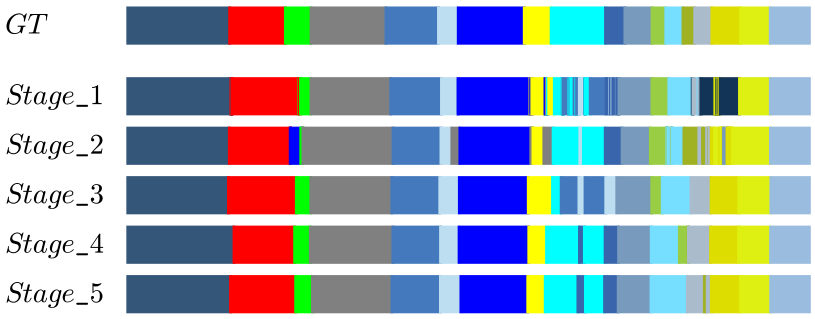


Figure 4.5: Qualitative result from the 50Salads dataset for comparing different number of stages.

shown in Figure 4.5. Adding more stages results in an incremental refinement of the predictions. In the rest of the experiments we use a multi-stage TCN with four stages.

Multi-Stage TCN vs. Deeper Single-Stage TCN

Table 4.1 showed that our multi-stage architecture is better than a single-stage one. However, this comparison does not show whether the improvement is because of the multi-stage architecture or due to the increase in the number of parameters when adding more stages. For a fair comparison, we train a single-stage model that has the same number of parameters as the multi-stage one. As each stage in our MS-TCN contains 12 layers (ten dilated convolutional layers, one 1×1 convolutional layer and a softmax layer), we train a single-stage TCN with 48 layers, which is the number of layers in a MS-TCN with four stages. For the dilated convolutions, we use similar dilation factors as in our MS-TCN. *I.e.* we start with a dilation factor of 1 and double it at every layer up to a factor of 512, and then we start again from 1. As shown in Table 4.2, our multi-stage architecture outperforms its single-stage counterpart with a large margin of up to 27%. This highlights the impact of the multi-stage architecture in improving the quality of the predictions.

	F1@{10,25,50}			Edit	Acc
SS-TCN (48 layers)	49.0	46.4	40.2	40.7	78.0
MS-TCN	**76.3**	**74.0**	**64.5**	**67.9**	**80.7**

Table 4.2: Comparing a multi-stage TCN with a deep single-stage TCN on the 50Salads dataset.

	F1@{10,25,50}			Edit	Acc
$\mathcal{L}_{cls}$	71.3	69.7	60.7	64.2	79.9
$\mathcal{L}_{cls} + \lambda\mathcal{L}_{KL}$	71.9	69.3	60.1	64.6	80.2
$\mathcal{L}_{cls} + \lambda\mathcal{L}_{T-MSE}$	**76.3**	**74.0**	**64.5**	**67.9**	**80.7**

Table 4.3: Comparing different loss functions on the 50Salads dataset.

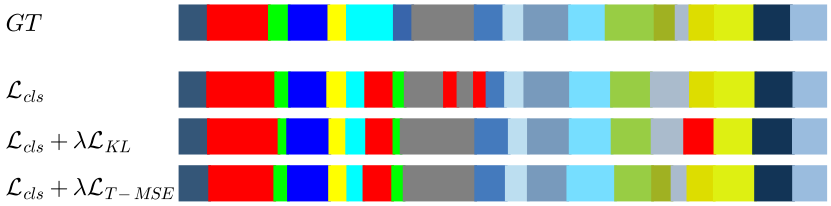


Figure 4.6: Qualitative result from the 50Salads dataset for comparing different loss functions.

4.3.2 Comparing Different Loss Functions

As a loss function, we use a combination of a cross-entropy loss, which is common practice for classification tasks, and a truncated mean squared loss over the frame-wise log-probabilities to ensure smooth predictions. While the smoothing loss slightly improves the frame-wise accuracy compared to the cross entropy loss alone, we found that this loss produces much less over-segmentation errors. Table 4.3 and Figure 4.6 show a comparison of these losses. As shown in Table 4.3, the proposed loss achieves better F1 and edit scores with an absolute improvement of up to 5%. This indicates that our loss produces less over-segmentation errors compared to cross entropy since it forces consecutive frames to have similar class probabilities, which results in a smoother output.

Penalizing the difference in log-probabilities is similar to the Kullback-Leibler (KL) divergence loss, which measures the difference between two probability distributions. However, the results show that the proposed loss produces better results than the KL loss as shown in Table 4.3 and Figure 4.6. The reason behind this is the fact that the KL divergence loss does not penalize cases where the difference between the target probability and the predicted probability is very small. Whereas the proposed loss penalizes small differences as well. Note that, in contrast to the KL loss, the proposed loss is symmetric. Figure 4.7 shows the surface for both the KL loss and the proposed truncated mean squared loss for the case of two classes. We also tried a symmetric version of the KL loss but it performed worse than the KL loss.

Impact of λ and τ

The effect of the proposed smoothing loss is controlled by two hyper-parameters: λ and τ. In this section, we study the impact of these parameters and see how they affect the performance of the proposed model.

Impact of λ: In all experiments, we set $\lambda = 0.15$. To analyze the effect of this parameter, we train different models with different values of λ. As shown in Table 4.4, the impact of λ is very small on

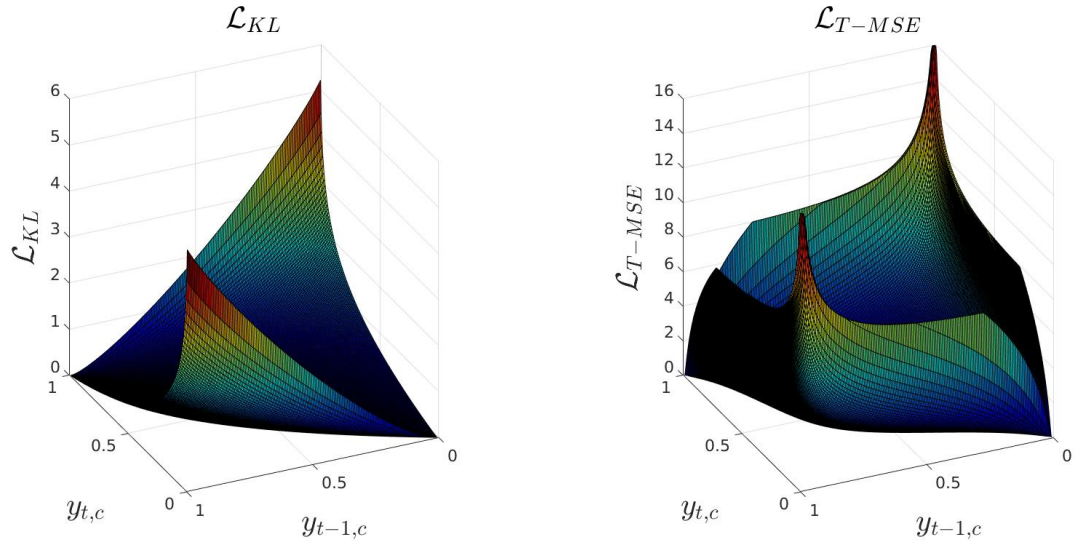


Figure 4.7: Loss surface for the Kullback-Leibler (KL) divergence loss ($\mathcal{L}_{KL}$) and the proposed truncated mean squared loss ($\mathcal{L}_{T-MSE}$) for the case of two classes. $y_{t,c}$ is the predicted probability for class c and $y_{t-1,c}$ is the target probability corresponding to that class.

the performance. Reducing λ to 0.05 still improves the performance but not as good as the default value of $\lambda = 0.15$. Increasing its value to $\lambda = 0.25$ also causes a degradation in performance. This drop in performance is due to the fact that the smoothing loss heavily penalizes changes in frame-wise labels, which affects the detected boundaries between action segments.

Impact of τ: This hyper-parameter defines the threshold to truncate the smoothing loss. Our default value is $\tau = 4$. While reducing the value to $\tau = 3$ still gives an improvement over the cross entropy baseline, setting $\tau = 5$ results in a huge drop in performance. This is mainly because when τ is too high, the smoothing loss penalizes cases where the model is very confident that the consecutive frames belong to two different classes, which indeed reduces the capability of the model in detecting the true boundaries between action segments.

Impact of λ	F1@{10,25,50}			Edit	Acc
MS-TCN ($\lambda = 0.05,\ \tau = 4$)	74.1	71.7	62.4	66.6	80.0
MS-TCN ($\lambda = 0.15,\ \tau = 4$)	**76.3**	**74.0**	**64.5**	67.9	**80.7**
MS-TCN ($\lambda = 0.25,\ \tau = 4$)	74.7	72.4	63.7	**68.1**	78.9
Impact of τ	F1@{10,25,50}			Edit	Acc
MS-TCN ($\lambda = 0.15,\ \tau = 3$)	74.2	72.1	62.2	67.1	79.4
MS-TCN ($\lambda = 0.15,\ \tau = 4$)	**76.3**	**74.0**	**64.5**	**67.9**	**80.7**
MS-TCN ($\lambda = 0.15,\ \tau = 5$)	66.6	63.7	54.7	60.0	74.0

Table 4.4: Impact of λ and τ on the 50Salads dataset.

	F1@{10,25,50}			Edit	Acc
Probabilities and features	56.2	53.7	45.8	47.6	76.8
Probabilities only	**76.3**	**74.0**	**64.5**	**67.9**	**80.7**

Table 4.5: Effect of passing features to higher stages on the 50Salads dataset.

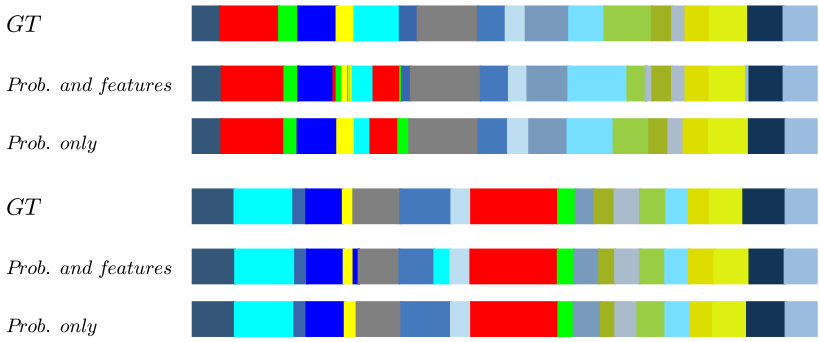


Figure 4.8: Qualitative results for two videos from the 50Salads dataset showing the effect of passing features to higher stages.

4.3.3 Effect of Passing Features to Higher Stages

In the proposed multi-stage TCN, the input to higher stages are the frame-wise probabilities only. However, in the multi-stage architectures that are used for human pose estimation, additional features are usually concatenated to the output heat-maps of the previous stage. In this experiment, we therefore analyze the effect of combining additional features to the input probabilities of higher stages. To this end, we trained two multi-stage TCNs: one only with the predicted frame-wise probabilities as input to the next stage, and, for the second model, we concatenated the output of the last dilated convolutional layer in each stage to the input probabilities of the next stage. As shown in Table 4.5, concatenating the features to the input probabilities results in a huge drop of the F1 score and the segmental edit distance (around 20%). We argue that the reason behind this degradation in performance is that a lot of action classes share similar appearance and motion. By adding the features of such classes at each stage, the model is confused and produces small separated falsely detected action segments that correspond to an over-segmentation effect. Passing only the probabilities forces the model to focus on the context of neighboring labels, which are explicitly represented by the probabilities. This effect can also be seen in the qualitative results shown in Figure 4.8.

4.3.4 MS-TCN++ vs. MS-TCN

In this section, we compare the two multi-stage architectures: MS-TCN++ and MS-TCN. In contrast to MS-TCN, MS-TCN++ uses the dual dilated layer (DDL) in the first stage. Table 4.6 shows the results of both architectures on the 50Salads dataset. As shown in the table, MS-TCN++ outperforms MS-TCN with a large margin of up to 6.4%. This emphasizes the importance of combining both local and global representations in the prediction generation stage by utilizing the DDL in MS-TCN++.

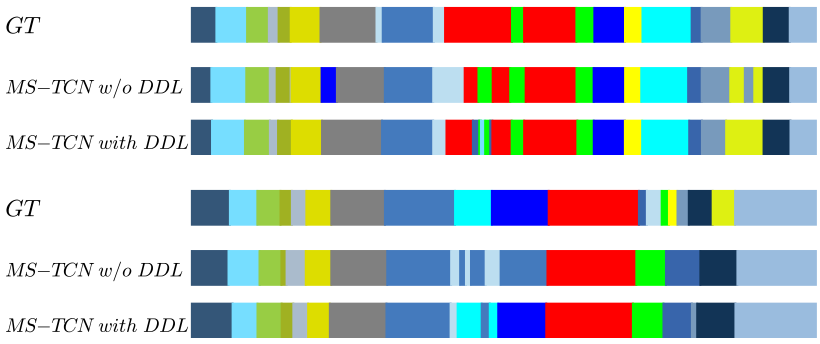


Figure 4.9: Qualitative results for two videos from the 50Salads dataset showing the impact of the dual dilated layer (DDL).

	F1@{10,25,50}			Edit	Acc
MS-TCN	76.3	74.0	64.5	67.9	80.7
MS-TCN with DDL	77.3	75.0	67.3	69.8	82.4
MS-TCN++	**80.7**	**78.5**	**70.1**	**74.3**	**83.7**

Table 4.6: MS-TCN++ vs. MS-TCN vs. MS-TCN with DDL on the 50Salads dataset.

To study the impact of using DDL in all stages, we also train an MS-TCN where we use the DDL in all stages. As shown in Table 4.6, MS-TCN++ outperforms MS-TCN with DDL in all stages. This indicates that decoupling the design of the refinement stages and the prediction generation stage is crucial. While utilizing the global context by the DDL is crucial for the prediction generation stage, the refinement stages focus more on the local context. By adding DDL to the refinement stages, the accuracy even drops due to overfitting. Note that using DDL in all stages outperforms MS-TCN with up to 2.8%. This further highlights the gains of the DDL. The impact of DDL is also visible in the qualitative results shown in Figure 4.9.

4.3.5 Impact of the Number of Layers

For MS-TCN and the refinement stages in MS-TCN++, we fix the number of layers in each stage to 10 layers. Whereas for the prediction generation stage in MS-TCN++, we set the number of layers to 11. In this section, we study the impact of these parameters. Table 4.7 shows the impact of the number of layers (L) for the MS-TCN stages on the 50Salads dataset. Increasing L form 8 to 10 significantly improves the performance. This is due to the increase in the receptive field. Using more than 10 layers ($L = 11$, $L = 12$) does not improve the frame-wise accuracy but slightly increases the F1 scores. We also tried to change the number of layers only in the refinement stages in MS-TCN. As shown in Table 4.8, this does not have a significant impact and using 10 layers achieves the best performance. Also for the refinement stages in MS-TCN++, the number of layers L_r does not have a significant impact on the performance. To be consistent with MS-TCN, we set $L_r = 10$ which achieves a reasonable trade-off on performance with respect to all metrics as shown in Table 4.8.

	F1@{10,25,50}			Edit	Acc
$L = 6$	53.2	48.3	39.0	46.2	63.7
$L = 8$	66.4	63.7	52.8	60.1	73.9
$L = 10$	76.3	74.0	64.5	67.9	**80.7**
$L = 11$	76.7	74.2	65.5	**69.7**	80.4
$L = 12$	**77.8**	**75.2**	**66.9**	69.6	80.5

Table 4.7: Effect of the number of layers (L) in each stage of MS-TCN on the 50Salads dataset.

		F1@{10,25,50}			Edit	Acc
MS-TCN	$L_r = 6$	74.3	71.5	62.8	66.0	78.6
	$L_r = 8$	75.4	72.4	64.3	**68.0**	79.5
	$L_r = 10$	**76.3**	**74.0**	**64.5**	67.9	**80.7**
	$L_r = 11$	75.0	72.0	63.5	67.6	80.3
	$L_r = 12$	74.1	71.2	62.3	65.7	79.1
MS-TCN++	$L_r = 6$	78.2	75.6	67.7	69.6	82.3
	$L_r = 8$	**80.9**	78.2	**70.2**	73.4	82.9
	$L_r = 10$	80.7	**78.5**	70.1	**74.3**	**83.7**
	$L_r = 11$	80.5	78.3	70.0	72.6	83.4
	$L_r = 12$	79.4	76.9	69.2	71.3	83.5

Table 4.8: Effect of the number of layers (L_r) in each refinement stage on the 50Salads dataset.

A similar behavior can be observed in Table 4.9 for the number of layers L_g in the prediction generation stage with $L_g = 11$ achieving the best performance. Generally speaking, the number of layers in each stage has more impact in MS-TCN compared to MS-TCN++. As the main difference between these two models is the dual dilation layer (DDL) that is used in MS-TCN++, this indicates that the DDL can better capture both local and global features to generate much better predictions.

	F1@{10,25,50}			Edit	Acc
$L_g = 6$	74.3	71.6	63.5	67.8	78.5
$L_g = 8$	77.4	75.3	67.8	70.3	80.8
$L_g = 10$	79.8	77.9	**71.0**	72.5	83.1
$L_g = 11$	**80.7**	**78.5**	70.1	**74.3**	**83.7**
$L_g = 12$	78.9	76.6	67.6	70.8	83.2

Table 4.9: Effect of the number of layers (L_g) in the prediction generation stage for MS-TCN++ on the 50Salads dataset.

	Duration	F1@{10,25,50}			Edit	Acc
MS-TCN	< 1 min	89.6	87.9	77.0	82.5	76.6
	$1 - 1.5$ min	85.9	84.3	71.9	80.7	76.4
	$\geqslant 1.5$ min	81.2	76.5	58.4	71.8	75.9
MS-TCN++	< 1 min	90.4	90.4	80.8	84.4	79.3
	$1 - 1.5$ min	88.7	85.8	75.1	83.6	79.3
	$\geqslant 1.5$ min	80.8	78.8	63.3	76.1	77.2

Table 4.10: Evaluation of three groups of videos from the GTEA dataset based on their durations.

	N_r	F1@{10,25,50}			Edit	Acc
MS-TCN++	0	51.0	48.4	40.7	40.4	80.7
MS-TCN++	1	70.7	68.2	59.7	62.0	82.4
MS-TCN++	2	77.8	75.1	66.9	69.4	82.5
MS-TCN++	3	**80.7**	78.5	**70.1**	**74.3**	**83.7**
MS-TCN++	4	80.6	**78.7**	**70.1**	73.1	82.4

Table 4.11: Impact of the number of refinement stages on the 50Salads dataset.

4.3.6 Impact of the Large Receptive Field on Short Videos

To study the impact of the large receptive field on short videos, we evaluate MS-TCN and MS-TCN++ on three groups of videos based on their durations. For this evaluation, we use the GTEA dataset since it contains shorter videos compared to the other datasets. As shown in Table 4.10, both MS-TCN and MS-TCN++ perform well on both short and long videos. Nevertheless, the performance is slightly worse on longer videos due to the limited receptive field. The improvements of MS-TCN++ over MS-TCN are also noticeable for both short and long videos.

4.3.7 Effect of the Number of Refinement Stages

We set the number of refinement stages N_r in MS-TCN++ to 3 stages, which results in a model with 4 stages in total. Table 4.11 shows the impact of the refinement stages on the 50Salads dataset. Using only the prediction generation stage ($N_r = 0$) results in a relative low performance but it is much better than a single stage TCN (Table 4.1). Adding more refinement stages improves the performance incrementally. Nevertheless, adding more than 3 refinement stages does not provide additional improvements.

4.3.8 Impact of Parameters Sharing

MS-TCN++ consists of a prediction generation stage and 3 refinement stages. Although adding more stages results in a better performance, it also increases the number of parameters. As the refinement

	F1@{10,25,50}			Edit	Acc	# param.(m)
MS-TCN	76.3	74.0	64.5	67.9	80.7	0.80
MS-TCN++	**80.7**	**78.5**	**70.1**	**74.3**	**83.7**	0.99
MS-TCN++(sh)	78.7	76.6	68.3	70.7	82.2	0.66

Table 4.12: Impact of sharing parameters for the refinement stages on the 50Salads dataset.

stages share in principle the same task, it is hence intuitive that they can share parameters. Table 4.12 shows the impact of sharing parameters between the refinement stages. Sharing parameters significantly reduces the number of parameters with only a slight decrease in performance. For an MS-TCN++ with 3 refinement stages, sharing parameters reduces the total number of parameters to roughly 66% of the total parameters in the original model. As shown in the table, MS-TCN++ with shared parameters outperforms MS-TCN with a margin of up to 3.8% despite having less parameters.

4.3.9 Impact of Temporal Resolution

Previous temporal models operate on a low temporal resolution of 1-3 frames per second (*Lea et al.*, 2017; *Lei and Todorovic*, 2018; *Ding and Xu*, 2018). On the contrary, our approach is able to handle a higher resolution of 15 fps. In this experiment, we evaluate MS-TCN and MS-TCN++, with and without parameter sharing, for a low temporal resolution of 1 fps. As shown in Table 4.13, both models are able to handle both low and high temporal resolutions. While reducing the temporal resolution for MS-TCN results in a better edit distance and segmental F1 score, using a higher resolution gives better frame-wise accuracy. Operating on a low temporal resolution makes MS-TCN less prone to the over-segmentation problem, which is reflected in the better edit and F1 scores. Due to the dual dilated layers, MS-TCN++ benefits more from a higher temporal resolution and the impact of reducing the temporal resolution for MS-TCN++ is noticeable for all evaluation metrics. Note that even when we share the parameters of the refinement stages in MS-TCN++, using a higher temporal resolution results in a better performance.

	F1@{10,25,50}			Edit	Acc
MS-TCN (1 fps)	**77.8**	**74.9**	64.0	**70.7**	78.6
MS-TCN (15 fps)	76.3	74.0	**64.5**	67.9	**80.7**
MS-TCN++ (1 fps)	80.4	**78.7**	68.6	73.3	81.1
MS-TCN++ (15 fps)	**80.7**	78.5	**70.1**	**74.3**	**83.7**
MS-TCN++(sh) (1 fps)	77.0	73.8	64.0	69.1	80.8
MS-TCN++(sh) (15 fps)	**78.7**	**76.6**	**68.3**	**70.7**	**82.2**

Table 4.13: Impact of temporal resolution on the 50Salads dataset.

		F1@{10,25,50}			Edit	Acc
w/o FT	SS-TCN	62.8	60.0	48.1	55.0	73.3
	MS-TCN	85.8	83.4	69.8	79.0	76.3
	MS-TCN++	87.0	85.2	73.5	82.0	78.7
	MS-TCN++(sh)	**87.8**	**86.2**	**74.4**	**82.6**	**78.9**
with FT	SS-TCN	69.5	64.9	55.8	61.1	75.3
	MS-TCN	87.5	85.4	74.6	81.4	79.2
	MS-TCN++	**88.8**	85.7	**76.0**	**83.5**	**80.1**
	MS-TCN++(sh)	88.2	**86.2**	75.9	83.0	79.7

Table 4.14: Effect of fine-tuning on the GTEA dataset.

4.3.10 Impact of Fine-tuning the Features

In our experiments, we use the I3D features without fine-tuning. Table 4.14 shows the effect of fine-tuning on the GTEA dataset. Both of our multi-stage architectures, MS-TCN and MS-TCN++, significantly outperform the single stage architecture - with and without fine-tuning. This also holds when the parameters of the refinement stages in MS-TCN++ are shared. Fine-tuning improves the results, but the effect of fine-tuning for action segmentation is lower than for action recognition. This is expected since the temporal model is by far more important for segmentation than for recognition.

Note that without fine-tuning, sharing parameters achieves better results on GTEA. This is mainly due to the reduced number of parameters, which prevents the model from over-fitting the training data, especially for small datasets like GTEA.

4.3.11 Comparison with the State-of-the-Art

We divide state-of-the-art methods into two groups: methods published before the proposal of our temporal convolutional networks, and methods that extend and build on our models. In the following we compare our models with both types of the state-of-the-art methods.

Comparison with Previous Approaches

In this section, we compare the proposed models to the state-of-the-art methods on three datasets: 50Salads, Georgia Tech Egocentric Activities (GTEA), and the Breakfast datasets. The results are presented in Table 4.15. As shown in the table, our models outperform the state-of-the-art methods on the three datasets and with respect to three evaluation metrics: F1 score, segmental edit distance, and frame-wise accuracy (Acc) with a large margin (up to 11.6% for the frame-wise accuracy on the 50Salads dataset). Qualitative results on the three datasets are shown in Figure 4.10. Note that all the reported results are obtained using the I3D features. To analyze the effect of using a different type of features, we evaluated MS-TCN on the Breakfast dataset using the improved dense trajectories (IDT) features, which are the commonly used features for the Breakfast dataset. As shown in Table 4.15, the impact of the features is very small. While the frame-wise accuracy and edit distance are slightly better using the I3D features, the model achieves a better F1 score when using the IDT features

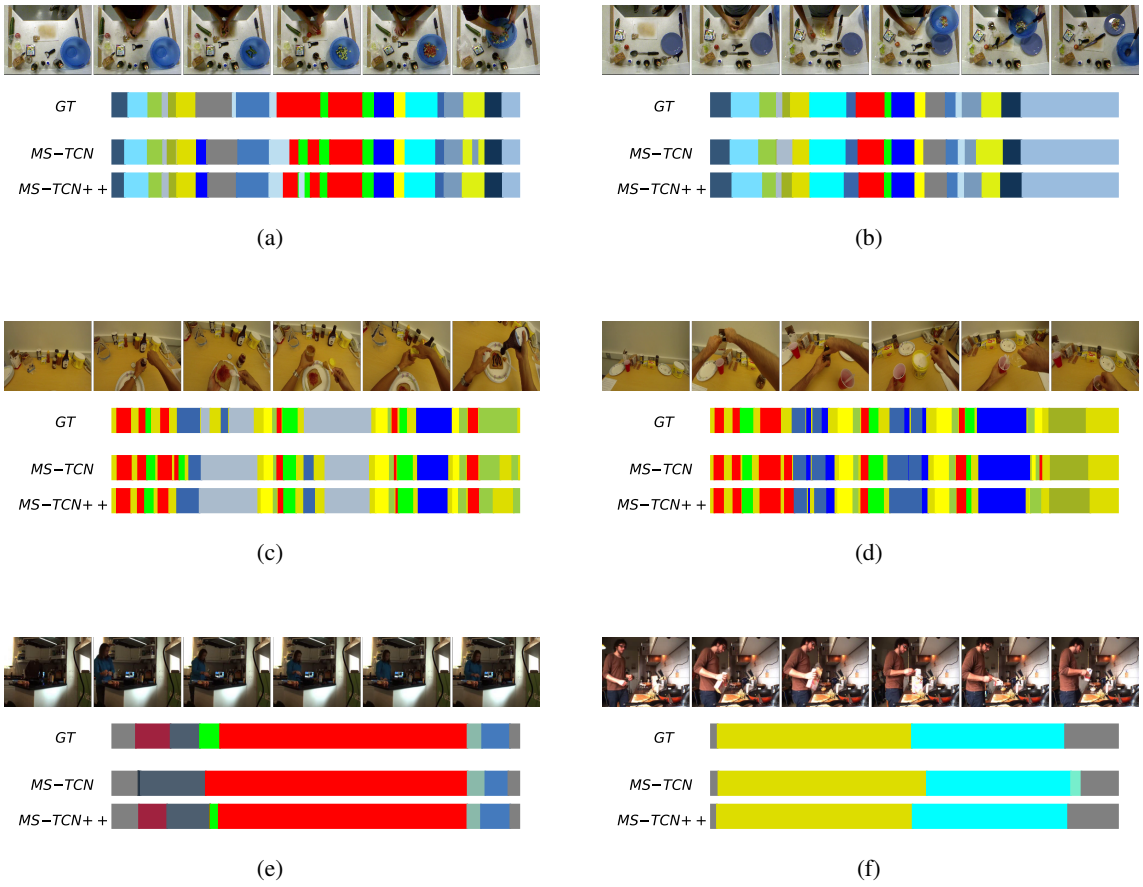


Figure 4.10: Qualitative results for the temporal action segmentation task on (a)(b) 50Salads, (c)(d) GTEA, and (e)(f) Breakfast dataset.

compared to I3D. This is mainly because I3D features encode both motion and appearance, whereas the IDT features encode only motion. For datasets like Breakfast, using appearance information does not help the performance since the appearance does not give a strong evidence about the action that is carried out. This can be seen in the qualitative results shown in Figure 4.10. The video frames share a very similar appearance. Additional appearance features therefore do not help in recognizing the activity. As shown in Table 4.15, sharing the parameters of the refinement stages achieves similar performance to MS-TCN++, but it requires about 66% less parameters as reported in Table 4.12.

As our approaches do not use any recurrent layers, they are very fast both during training and testing. Training MS-TCN++ for 50 epochs takes only 10 minutes on the 50Salads dataset compared to 35 minutes for training a single cell of a Bi-LSTM with a 64-dimensional hidden state on a single GTX 1080 Ti GPU. This is due to the sequential prediction of the LSTM, where the activations at any time step depend on the activations from the previous steps. For the MS-TCN and MS-TCN++, activations at all time steps are computed in parallel.

Comparison with Approaches Extending our Models

Motivated by the success of the proposed architectures, several approaches built on top of them and further improved the results. For instance, *Huang et al.* (2020) used the predictions of our models to construct a graph and then apply a graph convolutional network to model relations between the action segments. Other approaches predicted action boundaries and used them to aggregate predictions inside the action segments (*Wang et al.*, 2020; *Ishikawa et al.*, 2021). Recently, *Gao et al.* (2021) proposed a global-to-local search scheme to determine the dilation factors for each layer. *Souri et al.* (2021) used a TCN-based backbone similar to our models and combine it with a sequence-to-sequence model that predicts the action segments and their durations. A comparison with these approaches is provided in Table 4.16. Our models are still competitive with these approaches. Our architectures have also been used as a backbone for self-supervised domain adaption approaches for the action segmentation task (*Chen et al.*, 2020).

4.4 Summary

We presented two multi-stage architectures for the temporal action segmentation task. While the first stage generates an initial prediction, this prediction is iteratively refined by the higher stages. Instead of the commonly used temporal pooling, we used dilated convolutions to increase the temporal receptive field. The experimental evaluation demonstrated the capability of our architecture in capturing temporal dependencies between action classes and reducing over-segmentation errors. We further introduced a smoothing loss that gives an additional improvement of the predictions quality. We also introduced a dual dilated layer that captures both local and global features, which results in better performance. Moreover, we showed that sharing the parameters in the refinement stages results in a more efficient model with a slight degradation in performance. Our models outperformed the state-of-the-art methods when the work has been published on three challenging datasets recorded from different views with a large margin. Since our models are fully convolutional, they are very efficient and fast both during training and testing.

50Salads	F1@{10,25,50}			Edit	Acc
Spatial CNN(*Lea et al.*, 2016)	32.3	27.1	18.9	24.8	54.9
IDT+LM(*Richard and Gall*, 2016)	44.4	38.9	27.8	45.8	48.7
Bi-LSTM(*Singh et al.*, 2016a)	62.6	58.3	47.0	55.6	55.7
Dilated TCN(*Lea et al.*, 2017)	52.2	47.6	37.4	43.1	59.3
ST-CNN(*Lea et al.*, 2016)	55.9	49.6	37.1	45.9	59.4
ED-TCN(*Lea et al.*, 2017)	68.0	63.9	52.6	52.6	64.7
TRN(*Lei and Todorovic*, 2018)	70.2	65.4	56.3	63.7	66.9
TDRN+UNet(*Lei and Todorovic*, 2018)	69.6	65.0	53.6	62.2	66.1
TDRN(*Lei and Todorovic*, 2018)	72.9	68.5	57.2	66.0	68.1
LCDC+ED-TCN(*Mac et al.*, 2019)	73.8	-	-	66.9	72.1
MS-TCN	76.3	74.0	64.5	67.9	80.7
MS-TCN++(sh)	78.7	76.6	68.3	70.7	82.2
MS-TCN++	**80.7**	**78.5**	**70.1**	**74.3**	**83.7**

GTEA	F1@{10,25,50}			Edit	Acc
Spatial CNN(*Lea et al.*, 2016)	41.8	36.0	25.1	-	54.1
Bi-LSTM(*Singh et al.*, 2016a)	66.5	59.0	43.6	-	55.5
Dilated TCN(*Lea et al.*, 2017)	58.8	52.2	42.2	-	58.3
ST-CNN(*Lea et al.*, 2016)	58.7	54.4	41.9	-	60.6
ED-TCN(*Lea et al.*, 2017)	72.2	69.3	56.0	-	64.0
LCDC+ED-TCN(*Mac et al.*, 2019)	75.4	-	-	72.8	65.3
TRN(*Lei and Todorovic*, 2018)	77.4	71.3	59.1	72.2	67.8
TDRN+UNet(*Lei and Todorovic*, 2018)	78.1	73.8	62.2	73.7	69.3
TDRN(*Lei and Todorovic*, 2018)	79.2	74.4	62.7	74.1	70.1
MS-TCN	87.5	85.4	74.6	81.4	79.2
MS-TCN++(sh)	88.2	**86.2**	75.9	83.0	79.7
MS-TCN++	**88.8**	85.7	**76.0**	**83.5**	**80.1**

Breakfast	F1@{10,25,50}			Edit	Acc
ED-TCN (*Lea et al.*, 2017)*	-	-	-	-	43.3
HTK (*Kuehne et al.*, 2017)	-	-	-	-	50.7
TCFPN (*Ding and Xu*, 2018)	-	-	-	-	52.0
HTK(64) (*Kuehne et al.*, 2016)	-	-	-	-	56.3
GRU (*Richard et al.*, 2017)*	-	-	-	-	60.6
GRU+length prior (*Kuehne et al.*, 2020)	-	-	-	-	61.3
MS-TCN (IDT)	58.2	52.9	40.8	61.4	65.1
MS-TCN (I3D)	52.6	48.1	37.9	61.7	66.3
MS-TCN++(I3D) (sh)	63.3	57.7	44.5	64.9	67.3
MS-TCN++ (I3D)	**64.1**	**58.6**	**45.9**	**65.6**	**67.6**

Table 4.15: Comparison with the state-of-the-art on 50Salads, GTEA, and the Breakfast dataset. (* obtained from (*Ding and Xu*, 2018)).

50Salads	F1@{10,25,50}			Edit	Acc
MS-TCN	76.3	74.0	64.5	67.9	80.7
MS-TCN++(sh)	78.7	76.6	68.3	70.7	82.2
MS-TCN++	80.7	78.5	70.1	74.3	83.7
MS-TCN + GTRM (*Huang et al.*, 2020)	75.4	72.8	63.9	67.5	82.6
BCN (*Wang et al.*, 2020)	82.3	81.3	74.0	74.3	84.4
ASRF (*Ishikawa et al.*, 2021)	84.9	83.5	77.3	79.3	84.5
MS-TCN + Global2Local (*Gao et al.*, 2021)	80.3	78.0	69.8	73.4	82.2
GTEA	F1@{10,25,50}			Edit	Acc
MS-TCN	87.5	85.4	74.6	81.4	79.2
MS-TCN++(sh)	88.2	86.2	75.9	83.0	79.7
MS-TCN++	88.8	85.7	76.0	83.5	80.1
BCN (*Wang et al.*, 2020)	88.5	87.1	77.3	84.4	79.8
ASRF (*Ishikawa et al.*, 2021)	89.4	87.8	79.8	83.7	77.3
MS-TCN + Global2Local (*Gao et al.*, 2021)	89.9	87.3	75.8	84.6	78.5
Breakfast	F1@{10,25,50}			Edit	Acc
MS-TCN (IDT)	58.2	52.9	40.8	61.4	65.1
MS-TCN (I3D)	52.6	48.1	37.9	61.7	66.3
MS-TCN++(I3D) (sh)	63.3	57.7	44.5	64.9	67.3
MS-TCN++ (I3D)	64.1	58.6	45.9	65.6	67.6
MS-TCN + GTRM (*Huang et al.*, 2020)	57.5	54.0	43.3	58.7	65.0
BCN (*Wang et al.*, 2020)	68.7	65.5	55.0	66.2	70.4
ASRF (*Ishikawa et al.*, 2021)	74.3	68.9	56.1	72.4	67.6
MuCon (*Souri et al.*, 2021)	73.2	66.1	48.4	76.3	62.8
MS-TCN + Global2Local (*Gao et al.*, 2021)	74.9	69.0	55.2	73.3	70.7

Table 4.16: Comparison with the state-of-the-art approaches extending our models on 50Salads, GTEA, and the Breakfast dataset.

CHAPTER 5

Temporal Action Segmentation from Timestamp Supervision

In the previous chapter, we proposed two temporal convolutional networks for action segmentation. While these networks show impressive performance in segmenting actions, training these models is expensive as it requires annotating every single frame in the training videos. Recently, several approaches managed to reduce the amount of required annotations. For instance, it is possible now to train an action segmentation model using only an ordered list of actions that occur in the video. However, the performance of these approaches is still significantly worse than the performance of the approaches that rely on the full supervision.

To alleviate the limitations of the previous approaches, we propose in this chapter to use timestamp supervision for the temporal action segmentation task. Timestamps require a comparable annotation effort to annotating the list of actions occurring in the video, and yet provide a more supervisory signal. To demonstrate the effectiveness of timestamp supervision, we propose an approach to train a segmentation model using only timestamps annotations. Our approach uses the model output and the annotated timestamps to generate frame-wise labels by detecting the action changes. We further introduce a confidence loss that forces the predicted probabilities to monotonically decrease as the distance to the timestamps increases. This ensures that all and not only the most distinctive frames of an action are learned during training. The evaluation on four datasets shows that models trained with timestamps annotations achieve comparable performance to the performance achieved by fully supervised approaches.

This chapter is based on (*Li et al.*, 2021). Yazan Abu Farha proposed the timestamp supervison for action segmentation, the confidence loss, designed the baselines, and wrote the paper. Zhe Li proposed the action change detection approach and ran all the experiments.

Contents

5.1 Introduction

Analyzing and understanding video content is very important for many applications, such as surveillance or intelligent advertisement. In the previous chapter, we proposed two architectures that have been very successful in analyzing and segmenting activities in videos. Despite the success of the proposed approaches, they rely on fully annotated videos where the start and end frames of each action segment are annotated.

This level of supervision, however, is very time consuming and hard to obtain. Furthermore, as the boundaries between action segments are usually ambiguous, this might result in inconsistencies between annotations obtained from different annotators. To alleviate these problems, many researchers start exploring weaker levels of supervision in the form of transcripts (*Bojanowski et al.*, 2014; *Richard et al.*, 2018b; *Li et al.*, 2019) or even sets (*Richard et al.*, 2018a; *Fayyaz and Gall*, 2020; *Li and Todorovic*, 2020). For transcript-level supervision, the videos are annotated with an ordered list of actions occurring in the video without the starting and ending time of each action. Whereas for the set-level supervision, only the set of actions are provided without any information regarding the order or how many times each action occurs in the videos.

While transcript-level and set-level supervision significantly reduce the annotation effort, the performance is not satisfying and there is still a gap compared to fully supervised approaches. In this chapter, inspired by the recently introduced timestamp supervision for action recognition (*Moltisanti et al.*, 2019), we propose to use timestamp supervision for the action segmentation task to address the limitations of the current weakly supervised approaches. For timestamp supervision, only one frame is annotated from each action segment as illustrated in Figure 5.1. Such timestamps annotations can be obtained with comparable effort to transcripts, and yet they provide more supervision. Besides the ordered list of actions occurring in the video, timestamps annotations give partial information about the location of the action segments, which can be utilized to further improve the performance.

Given the timestamps annotations, the question is how to train a segmentation model with such level of supervision. A naive approach takes only the sparsely annotated frames for training. This, however, ignores most of the information in the video and does not achieve good results as we will show in the experiments. Another strategy is to iterate the process and consider frames with high confidence scores near the annotations as additional annotated frames and include them during training (*Moltisanti et al.*, 2019). Furthermore, frames that are far away from the annotations can be considered as negative samples (*Ma et al.*, 2020). For temporal action segmentation, which is comparable to image semantic segmentation, however, all frames need to be annotated and there are no large parts of the video that can be used to sample negative examples. Furthermore, relying only on frames with high confidence discards many of the video frames that occur during an action and focuses only on the most distinctive frames of an action, which can be sufficient for action recognition or detection but not for action segmentation.

In this chapter, we therefore propose a different approach where all frames of the videos are used. Instead of detecting frames of high confidences, we aim to identify changes of actions in order to divide the videos into segments. Since for each action change the frames before the change

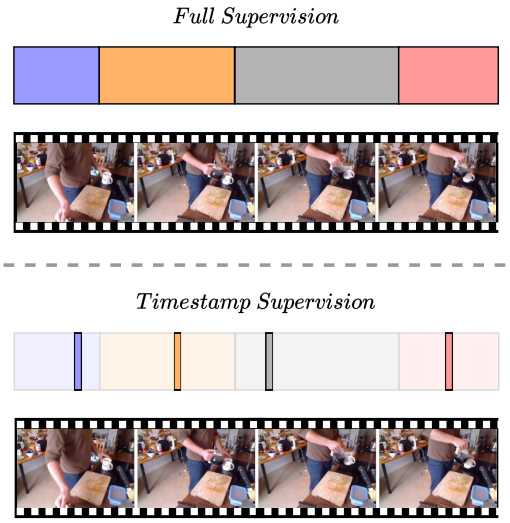


Figure 5.1: For fully supervised temporal action segmentation, each frame in the training videos is annotated with an action label (top). This process is time-consuming since it requires an accurate annotation of the start and end frame of each action. To reduce the annotation effort, we propose to use timestamps as supervision (bottom). In this case, only one arbitrary frame needs to be annotated for each action and the annotators do not need to search for the start and end frames, which is the most time-consuming annotation part.

should be assigned to the previous timestamp and after the change to the next timestamp, we find the action changes by minimizing the variations of the features within each of the two clusters of frames. While we can then train the model on all frames by assigning the label of the timestamp to the corresponding frames, it does not guarantee that all frames of an action are effectively used. We therefore introduce a loss function that enforces a monotonic decrease in the class probabilities as the distance to the timestamps increases. This loss encourages the model to predict higher probabilities for low confident regions that are surrounded by high confident frames and therefore to use all frames and not only the most distinctive frames.

Our contribution is thus three folded.

1. We propose to use timestamp supervision for the temporal action segmentation task, where the goal is to predict frame-wise action labels for untrimmed videos.

2. We introduce an approach to train a temporal action segmentation model from timestamp supervision. The approach uses the model predictions and the annotated timestamps for estimating action changes.

3. We propose a novel confidence loss that forces the model confidence to decrease monotonically as the distance to the timestamp increases.

We evaluate our approach on four datasets: 50Salads (*Stein and McKenna*, 2013), Breakfast (*Kuehne et al.*, 2014), BEOID (*Damen et al.*, 2014), and Georgia Tech Egocentric Activities (GTEA) (*Fathi et al.*, 2011b). We show that training an action segmentation model is feasible with only timestamp supervision without compromising the performance compared to the fully supervised

approaches. On the 50Salads dataset, for instance, we achieve 97% of the accuracy compared to fully supervised learning, but at a tiny fraction of the annotation costs.

5.2 Temporal Action Segmentation

Temporal action segmentation is the task of predicting frame-wise action labels for a given input video. Formally, given a sequence of video frames $x_{1:T} = (x_1, \dots, x_T)$, where T is the number of frames, the goal is to predict a sequence of frame-wise action labels $a_{1:T} = (a_1, \dots, a_T)$. In contrast to the fully supervised approaches, which assume that the frame-wise labels are given at training time, we consider a weaker level of supervision in the form of timestamps. In Section 5.2.1 we introduce the timestamp supervision for the temporal action segmentation task. Then, we describe the proposed framework for learning from timestamp supervision in Section 5.2.2. Finally, we provide the details of the loss function in Section 5.2.3.

5.2.1 Timestamp Supervision

In a fully supervised setup, the frame-wise labels $(a_1, \dots, a_T)$ of the training videos are available. On the contrary, for timestamp supervision, only a single frame for each action segment is annotated. Given a training video with T frames and N action segments, where $N << T$, only N frames are annotated with labels $A_{TS} = [a_{t_1}, \dots, a_{t_N}]$, where frame t_i belongs to the $i\text{-}th$ action segment. To annotate timestamps, one can go fast forward through a video and press a button when an action occurs. This does not take more time than annotating transcripts. Whereas annotating the start and end frames of each action requires going slowly back and forth between the frames. As reported in (*Ma et al.*, 2020), annotators need 6 times longer to annotate the start and end frame compared to annotating a single timestamp. While timestamps are much easier to obtain compared to the full annotation of the video frames, they provide much more information compared to weaker forms of supervision such as transcripts. Figure 5.1 illustrates the difference between timestamp supervision and full supervision.

5.2.2 Action Segmentation from Timestamp Supervision

Our goal is to train an action segmentation model $\mathcal{M}$ to predict action labels for each frame in the input video. If the frame-wise labels are available during the training, as in the fully supervised case, then it is possible to apply a classification loss on the output of the model $\mathcal{M}$ for each frame in the input video. However, in timestamp supervision, only a sparse set of frames are annotated. To alleviate this problem, we propose to generate frame-wise labels for the training videos, and use them as a target for the loss function as illustrated in Figure 5.2.

Detecting Action Changes. Given the timestamps annotations $A_{TS} = [a_{t_1}, \dots, a_{t_N}]$ for a training video, we want to generate frame-wise action labels $\hat{a}_{1:T} = (\hat{a}_1, \dots, \hat{a}_T)$ for each frame in that video such that $\hat{a}_{t_i} = a_{t_i}$ for $i \in [1, N]$. As for each action segment there is an annotated frame, finding the frame-wise labels can be reduced to finding the action change between each consecutive annotated timestamp. To this end, we pass the input video $x_{1:T}$ to the segmentation model $\mathcal{M}$, which will be described in Section 5.3, and use the output of the penultimate layer H combined with the timestamps

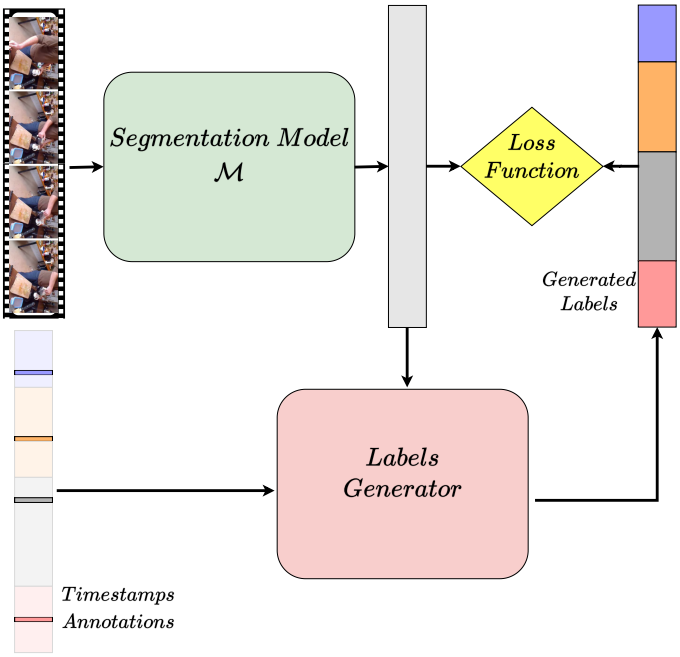


Figure 5.2: The framework of the proposed approach for training with timestamp supervision. Given the output of the segmentation model and the timestamps annotations, we generate action labels for each frame in the input video by estimating where the action labels change. A loss function is then computed between the predictions and the generated labels.

annotations to estimate where the action labels change between the timestamps. To generate the labels, all the frames that lie between an annotated timestamp and an estimated time of action change are assigned with the same action label as the annotated timestamp as illustrated in Figure 5.3. To detect the action change between two timestamps t_i and t_{i+1}, we find the time t_{b_i} that minimizes the following stamp-to-stamp energy function

$$t_{b_i} = \arg\min_{\hat{t}} \sum_{t=t_i}^{\hat{t}} d(h_t, c_i) + \sum_{t=\hat{t}+1}^{t_{i+1}} d(h_t, c_{i+1}),$$

$$s.t.$$

$$c_i = \frac{1}{\hat{t} - t_i} \sum_{t=t_i}^{\hat{t}} h_t, \tag{5.1}$$

$$c_{i+1} = \frac{1}{t_{i+1} - \hat{t} - 1} \sum_{t=\hat{t}+1}^{t_{i+1}} h_t,$$

$$t_i < \hat{t} < t_{i+1},$$

where $d(.,.)$ is the Euclidean distance, h_t is the output of the penultimate layer at time t, c_i is the average of the output between the first timestamp t_i and the estimate $\hat{t}$, and c_{i+1} is the average of the output between the estimate $\hat{t}$ and the second timestamp t_{i+1}. *I.e.*, we find the time t_{b_i} that divides the frames between two timestamps into two clusters with the minimum distance between frames and the corresponding cluster center.

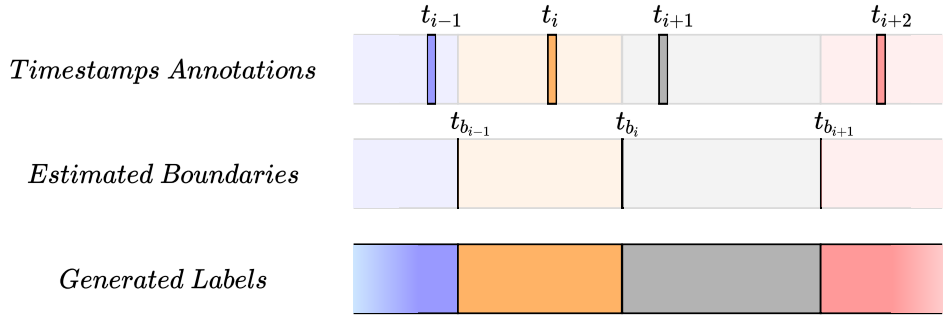


Figure 5.3: Given the timestamps annotations, we first estimate where the actions change between consecutive timestamps. To generate the frame-wise labels, all the frames that lie between an annotated timestamp and an estimated time of action change are assigned with the same action label as the annotated timestamp.

Forward-Backward Action Change Detection. In (5.1), the stamp-to-stamp energy function considers only the frames between the annotated timestamps to estimate where the actions change. Nonetheless, if we already have an estimate for $t_{b_{i-1}}$, then we already know that frames between the estimate $t_{b_{i-1}}$ and the next timestamp t_i will be assigned to action label a_{t_i}. This information can be used to estimate the time of action change for the next action segment t_{b_i}. The same argument also holds if we start estimating the boundaries in reverse order. *I.e.*, if we already know $t_{b_{i+1}}$, then frames between t_{i+1} and $t_{b_{i+1}}$ can be used to estimate t_{b_i}. We call the former estimate a forward estimate for the $i\text{-}th$ action change, whereas the later is called the backward estimate. The final estimate for t_{b_i} is the average of these two estimates. Formally

$$t_{b_i} = \frac{t_{b_i,FW} + t_{b_i,BW}}{2}$$

$$s.t.$$

$$t_{b_i,FW} = \arg\min_{\hat{t}} \sum_{t=t_{b_{i-1}}}^{\hat{t}} d(h_t, c_i) + \sum_{t=\hat{t}+1}^{t_{i+1}} d(h_t, c_{i+1}), \tag{5.2}$$

$$t_{b_i,BW} = \arg\min_{\hat{t}} \sum_{t=t_i}^{\hat{t}} d(h_t, c_i) + \sum_{t=\hat{t}+1}^{t_{b_{i+1}}} d(h_t, c_{i+1}).$$

5.2.3 Loss Function

For the fully supervised action segmentation models in Chapter 4, we used a combination of a classification loss and a smoothing loss. Besides these losses, we further introduce a novel confidence loss for the timestamp supervision. In the following, we describe in detail each loss function.

Classification Loss. We use a cross entropy loss between the predicted action probabilities and the corresponding generated target label

$$\mathcal{L}_{cls} = \frac{1}{T} \sum_t -log(y_{t,\hat{a}}), \tag{5.3}$$

where $y_{t,\hat{a}}$ is the predicted probability for the target label $\hat{a}$ at time t.

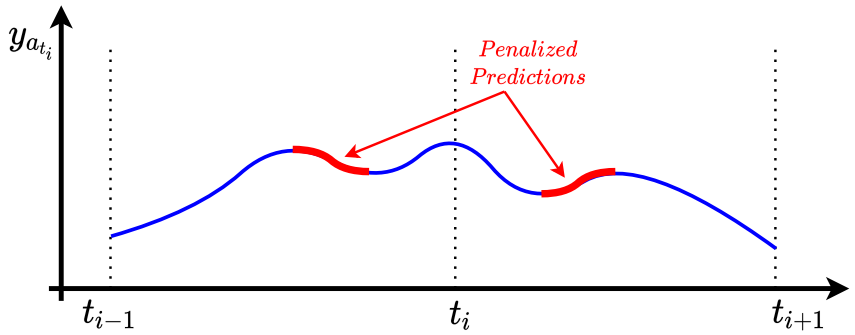


Figure 5.4: The confidence loss penalizes increases in the model confidence for label a_{t_i} as we move a way from the annotated timestamp t_i.

Smoothing Loss. As the classification loss treats each frame independently, it might result in an undesired over-segmentation effect. To encourage a smooth transition between frames and reduce over-segmentation errors, we use the truncated mean squared error proposed in Chapter 4

$$\mathcal{L}_{T-MSE} = \frac{1}{TC} \sum_{t,a} \tilde{\Delta}^2_{t,a}, \tag{5.4}$$

$$\tilde{\Delta}_{t,a} = \begin{cases} \Delta_{t,a} & : \Delta_{t,a} \leqslant \tau \\ \tau & : otherwise \end{cases}, \tag{5.5}$$

$$\Delta_{t,a} = |\log y_{t,a} - \log y_{t-1,a}|, \tag{5.6}$$

where T is the video length, C is the number of action classes, and $y_{t,a}$ is the probability of action a at time t.

Confidence Loss. Our approach relies on the model output to detect action changes. Nonetheless, as some frames are more informative than others, the model confidence might alternate between high and low values within the same action segment. Such behavior might result in ignoring regions with low confidence within the segments. To alleviate this problem, we apply the following loss

$$\mathcal{L}_{conf} = \frac{1}{T'} \sum_{a_{t_i} \in A_{TS}} \left(\sum_{t=t_{i-1}}^{t_{i+1}} \delta_{a_{t_i},t} \right), \tag{5.7}$$

$$\delta_{a_{t_i},t} = \begin{cases} max(0,\ \log y_{t,a_{t_i}} - \log y_{t-1,a_{t_i}}) & if\ t \geqslant t_i \\ max(0,\ \log y_{t-1,a_{t_i}} - \log y_{t,a_{t_i}}) & if\ t < t_i \end{cases}, \tag{5.8}$$

where $y_{t,a_{t_i}}$ is the probability of action a_{t_i} at time t, and $T' = 2(t_N - t_1)$ is the number of frames that contributed to the loss. For the first and last timestamps, we set $t_0 = t_1$ and $t_{N+1} = t_N$. This loss penalizes increase in confidence as we deviate from the annotations as illustrated in Figure 5.4.

Enforcing monotonicity on the model confidence has two effects as shown in Figure 5.6. First, it encourages the model to predict higher probabilities for low confident regions that are surrounded by regions with high confidence. Second, it suppresses outlier frames with high confidence that are far from the timestamps and not supported by high confident regions.

The final loss function to train the segmentation model is the sum of these three losses

$$\mathcal{L}_{total} = \mathcal{L}_{cls} + \alpha \mathcal{L}_{T-MSE} + \beta \mathcal{L}_{conf}, \tag{5.9}$$

where α and β are hyper-parameters to balance the contribution of each loss.

5.3 Experiments

Datasets. We evaluate our approach on four datasets: 50Salads (*Stein and McKenna*, 2013), Breakfast (*Kuehne et al.*, 2014), BEOID (*Damen et al.*, 2014), and Georgia Tech Egocentric Activities (GTEA) (*Fathi et al.*, 2011b). Details about the datasets are available in Chapter 2.

To generate the timestamps annotations, we randomly select one frame from each action segment in the training videos. We further evaluate our approach using human and noisy annotations in Section 5.3.6.

Metrics. We use the standard metrics for fully supervised action segmentation and report frame-wise accuracy (Acc), segmental edit score (Edit) and segmental F1 scores at overlapping thresholds 10%, 25% and 50%. Formal definitions for the evaluation metrics are discussed in Chapter 3.

Baselines. We implement two baselines: a Naive and a Uniform baseline. The Naive baseline computes the loss at the annotated timestamps only and does not generate frame-wise labels. Whereas the Uniform baseline generates the frame-wise labels by assuming that action labels change at the center frame between consecutive timestamps.

Implementation Details. We use a multi-stage temporal convolutional network (see Chapter 4) as a segmentation model $\mathcal{M}$. Following (*Vats et al.*, 2020), we use two parallel stages for the first stage with kernel size 5 and 3 and pass the sum of the outputs to next stages. We also experimented with different segmentation models in Section 5.3.4. We train our model for 50 epochs with Adam optimizer. To minimize the impact of initialization, only the annotated timestamps are used for the classification loss in the first 30 epochs, and the generated labels are used afterwards. The learning rate is set to 0.0005 and the batch size is 8. Following Chapter 4, we set $\tau = 4$ and $\alpha = 0.15$ for the smoothing loss and set $\beta = 0.075$. As input for our model, we use the I3D (*Carreira and Zisserman*, 2017) features of the video frames.

5.3.1 Comparison with the Baselines

In this section, we compare the proposed approach for action segmentation from timestamp supervision with the naive and uniform baselines. The results on the three datasets are shown in Table 5.1. Our approach outperforms these baselines with a large margin in all the evaluation metrics. While the naive baseline achieves a good frame-wise accuracy, it suffers from a severe over-segmentation problem as indicated by the low F1 and Edit scores. This is because it only uses the sparse timestamps annotations for training, which leaves a lot of ambiguity for frames without annotations. Using the uniform baseline reduces the over-segmentation by uniformly assigning a label for each frame. However, this results in inferior frame-wise accuracy as the uniform assignment generates many wrong labels. On the contrary, our approach utilizes the model predictions to generate much better target labels, which is reflected in the performance as illustrated in Figure 5.5. We also compare the performance of our approach to the fully supervised setup in Table 5.1. Our approach achieves comparable performance to the fully supervised case. Interestingly, our weakly supervised model achieves slightly higher F1 scores than the fully supervised model on the 50Salads dataset. This can be attributed to the ambiguity in the annotations. In contrast to timestamps, annotating the start and end of an action is ambiguous. Fully supervised approaches are therefore trained with slightly inconsistent annotations, which results in a few over-segmentation errors. The fully supervised setting has therefore always a higher frame accuracy, but the F1 score can be lower due to over-segmentation.

	F1@{10, 25, 50}			Edit	Acc
50Salads					
Naive	47.9	43.3	34.0	37.2	69.6
Uniform	62.9	58.2	42.3	60.4	63.4
Ours	**73.9**	**70.9**	**60.1**	**66.8**	**75.6**
Full Supervision	70.8	67.7	58.6	63.8	77.8
Breakfast					
Naive	34.1	29.1	20.1	37.4	56.8
Uniform	66.2	56.3	36.4	68.1	51.0
Ours	**70.5**	**63.6**	**47.4**	**69.9**	**64.1**
Full Supervision	69.9	64.2	51.5	69.4	68.0
GTEA					
Naive	59.7	55.3	39.6	51.1	56.5
Uniform	**78.9**	72.5	50.9	**73.1**	56.5
Ours	**78.9**	**73.0**	**55.4**	72.3	**66.4**
Full Supervision	85.1	82.7	69.6	79.6	76.1

Table 5.1: Comparison with the baselines on the three datasets.

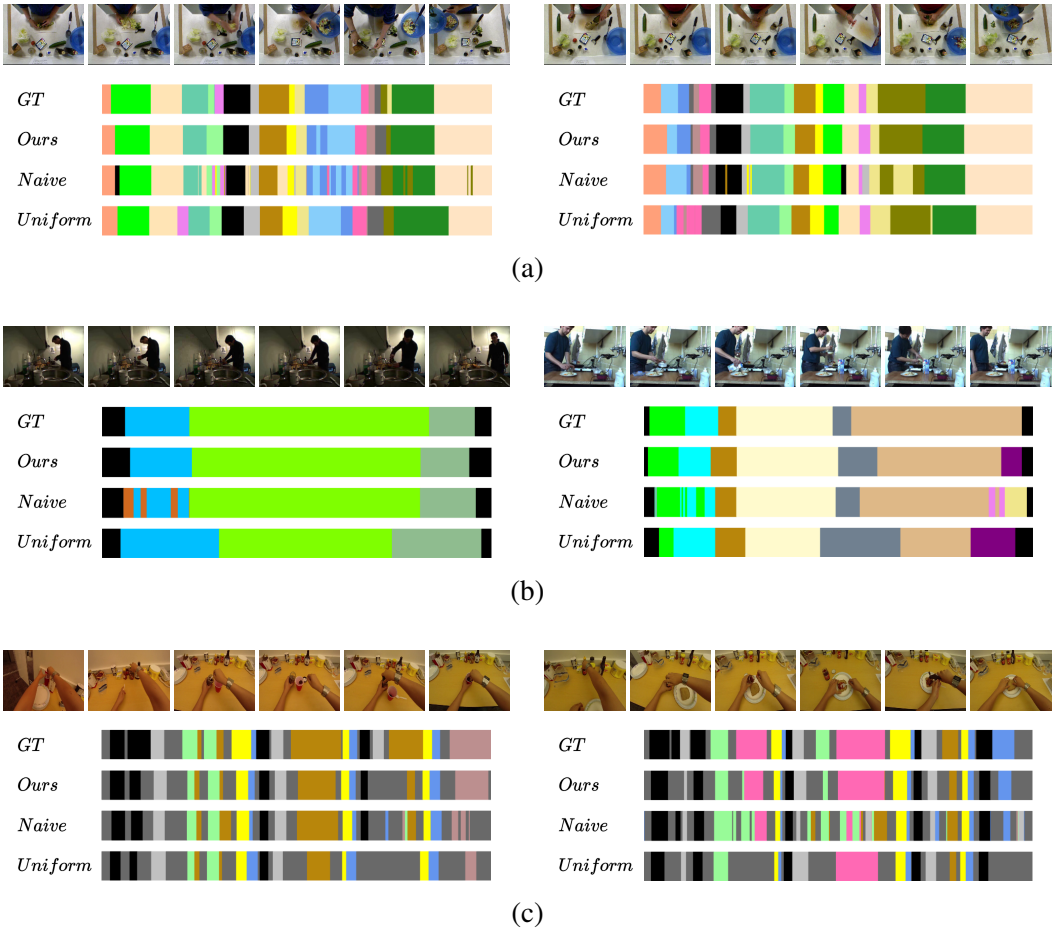


Figure 5.5: Qualitative results on (a) 50Salads, (b) Breakfast, and (c) GTEA datasets. As the naive baseline only trains on the sparse annotations, it suffers from an over-segmentation problem. While the uniform baseline reduces this problem by uniformly assigning labels to the frames, the durations of the predicted segments are not accurate and the predictions tend towards a uniform segmentation of the videos. On the contrary, our approach generates better predictions by utilizing the model output to detect where the action labels change.

	F1@{10, 25, 50}			Edit	Acc
50Salads					
$\mathcal{L}_{cls}$	65.7	62.6	50.7	57.7	72.8
$\mathcal{L}_{cls} + \alpha\mathcal{L}_{T-MSE}$	70.1	66.8	55.3	62.6	74.6
$\mathcal{L}_{cls} + \beta\mathcal{L}_{conf}$	73.2	70.6	60.1	65.2	75.3
$\mathcal{L}_{cls} + \alpha\mathcal{L}_{T-MSE} + \beta\mathcal{L}_{conf}$	**73.9**	**70.9**	**60.1**	**66.8**	**75.6**
Breakfast					
$\mathcal{L}_{cls}$	60.3	52.8	36.7	64.2	60.2
$\mathcal{L}_{cls} + \alpha\mathcal{L}_{T-MSE}$	67.5	60.1	44.3	68.9	63.7
$\mathcal{L}_{cls} + \beta\mathcal{L}_{conf}$	67.6	60.4	43.7	68.0	61.6
$\mathcal{L}_{cls} + \alpha\mathcal{L}_{T-MSE} + \beta\mathcal{L}_{conf}$	**70.5**	**63.6**	**47.4**	**69.9**	**64.1**

Table 5.2: Contribution of the different loss functions on the 50Salads and Breakfast datasets.

	F1@{10, 25, 50}			Edit	Acc
$\beta = 0$	70.1	66.8	55.3	62.6	74.6
$\beta = 0.025$	70.9	68.8	57.4	63.4	**76.2**
$\beta = 0.05$	73.1	70.2	58.7	65.4	75.6
$\beta = 0.075$	**73.9**	**70.9**	**60.1**	**66.8**	75.6
$\beta = 0.1$	73.2	70.6	**60.1**	66.1	74.6

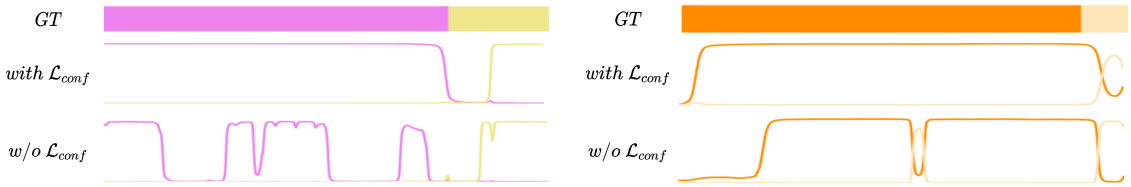
Table 5.3: Impact of β on the 50Salads dataset.

5.3.2 Impact of the Loss Function

The loss function to train our model consists of three losses: a classification loss, a smoothing loss, and a confidence loss. Table 5.2 shows the impact of each loss on both the 50Salads and the Breakfast dataset. While either of the smoothing loss and the confidence loss gives an additional boost in performance, the best performance is achieved when both of the losses are combined with the classification loss with a frame-wise accuracy improvement of 3.9% and 2.8% on 50Salads and the Breakfast dataset respectively, and roughly 10% on the F1 score at 50% overlapping threshold.

While the smoothing loss forces a smooth transition between consecutive frames, it does not take the annotations into account. On the contrary, the confidence loss forces the predicted probabilities to monotonically decrease as the distance to the timestamps increases. This encourages the model to have a high confidence for all frames within an action segment, and yet it suppresses outlier frames that are far from the annotations and not supported by regions with high confidence as illustrated in Figure 5.6.

To balance the contribution of the different losses, we set the weight of the smoothing loss to 0.15, and the weight of the confidence loss $\beta = 0.075$. In Table 5.3, we study the impact of β on the performance on the 50Salads dataset. As shown in the table, good results are achieved for β between 0.05 and 0.1.



(a) (b)

Figure 5.6: Impact of the confidence loss. Forcing monotonicity encourages the model to have a high confidence for all frames within an action segment (a). It also suppresses outlier frames with high confidence (b).

	F1@{10, 25, 50}			Edit	Acc
50Salads					
Stamp-to-Stamp (Prob.)	67.5	61.8	48.6	61.1	68.9
Stamp-to-Stamp (Features)	73.4	70.5	59.9	66.7	74.2
Ours	**73.9**	**70.9**	**60.1**	**66.8**	**75.6**
Breakfast					
Stamp-to-Stamp (Prob.)	65.7	55.9	35.9	68.0	58.8
Stamp-to-Stamp (Features)	66.3	59.6	44.4	67.9	60.1
Ours	**70.5**	**63.6**	**47.4**	**69.9**	**64.1**

Table 5.4: Impact of the energy function for action change detection on the 50Salads and Breakfast datasets.

5.3.3 Impact of the Energy Function for Action Change Detection

Our approach generates target labels by estimating where the action labels change using the forward-backward estimate as in (5.2). To analyze the impact of this estimate, we train another model that directly uses the stamp-to-stamp estimate (Stamp-to-Stamp (Features)) as in (5.1). As shown in Table 5.4, our approach performs better. We also tried another variant of the stamp-to-stamp energy function that maximizes the average probabilities of the action segments (Stamp-to-Stamp (Prob.)) instead of minimizing the distances to cluster centers. However, the performance is worse than the proposed energy function.

5.3.4 Impact of the Segmentation Model $\mathcal{M}$

In all experiments, we used a multi-stage temporal convolutional architecture based on Chapter 4 and (*Vats et al.*, 2020). In this section we study the impact of the segmentation model on the performance. To this end, we apply the proposed training scheme on the original MS-TCN and MS-TCN++ that we proposed in Chapter 4. As shown in Table 5.5, our approach is agnostic to the segmentation model and performs well with all these models.

Dataset	Seg. Model $\mathcal{M}$	F1@{10, 25, 50}			Edit	Acc
50Salads	MS-TCN (Chapter 4)	71.7	68.7	57.0	64.0	74.7
	MS-TCN++ (Chapter 4)	75.0	71.1	55.8	67.2	72.9
	Ours	73.9	70.9	60.1	66.8	75.6
GTEA	MS-TCN (Chapter 4)	79.8	73.3	47.7	76.3	59.3
	MS-TCN++ (Chapter 4)	78.3	72.2	49.1	74.5	62.2
	Ours	78.9	73.0	55.4	72.3	66.4

Table 5.5: Impact of the segmentation model $\mathcal{M}$ on the 50Salads and GTEA datasets.

5.3.5 Frame Selection for the Timestamp Annotations

The used datasets in this work are fully annotated. To simulate timestamp supervision, we randomly select one frame from each action segment in the training videos. To study the impact of the frame selection strategy, Table 5.6 shows results for two additional settings. While using the center frame of each segment achieves comparable results to a random frame, the performance drops when the start frame is used. Humans, however, would not annotate the start frame since it is more ambiguous (see Figure 4 in (*Ma et al.*, 2020)).

Timestamps	F1@{10, 25, 50}			Edit	Acc
Start frame	65.5	52.2	28.0	70.4	51.2
Center frame	**70.8**	63.5	45.4	**71.3**	61.3
Random	70.5	**63.6**	**47.4**	69.9	**64.1**

Table 5.6: Using start, center, or random frame as timestamp for each action on the Breakfast dataset.

5.3.6 Comparison with the State-of-the-Art

In this section, we compare our approach with recent state-of-the-art approaches for timestamp supervision. To the best of our knowledge, timestamp supervision has not been studied for the temporal action segmentation task. We, therefore, compare with similar methods in the context of action recognition (*Moltisanti et al.*, 2019) and action localization (*Ma et al.*, 2020).

Since the approach of (*Moltisanti et al.*, 2019) assumes the testing videos are trimmed and does not work for long untrimmed videos, we replaced their backbone network with our segmentation model for a fair comparison. To this end, we initialized the plateau functions around the timestamps annotations of the training videos and iteratively update their parameters based on the segmentation model output as in (*Moltisanti et al.*, 2019). Results on 50Salads, Breakfast, and GTEA datasets are shown in Tables 5.7-5.9, respectively. Our approach outperforms (*Moltisanti et al.*, 2019) on all datasets with a large margin of up to 13.5% frame-wise accuracy and 11.8% for the F1 score with 50% overlapping threshold on the GTEA dataset.

We also compare our approach for timestamp supervision with other levels of supervision for the temporal action segmentation task. As shown in Tables 5.7-5.9, timestamp supervision outperforms

Supervision	Method	F1@{10, 25, 50}			Edit	Acc
Full	MS-TCN (Chapter 4)	76.3	74.0	64.5	67.9	80.7
	MS-TCN++ (Chapter 4)	80.7	78.5	70.1	74.3	83.7
	BCN (*Wang et al.*, 2020)	82.3	81.3	74.0	74.3	84.4
	ASRF (*Ishikawa et al.*, 2021)	84.9	83.5	77.3	79.3	84.5
Timestamps	Seg. Model $\mathcal{M}$ + plateau (*Moltisanti et al.*, 2019)	71.2	68.2	56.1	62.6	73.9
	Ours	**73.9**	**70.9**	**60.1**	**66.8**	**75.6**
Transcripts	CDFL (*Li et al.*, 2019)	-	-	-	-	54.7
	NN-Viterbi (*Richard et al.*, 2018b)	-	-	-	-	49.4
	HMM-RNN (*Richard et al.*, 2017)	-	-	-	-	45.5

Table 5.7: Comparison with different levels of supervision on the 50Salads dataset.

Supervision	Method	F1@{10, 25, 50}			Edit	Acc
Full	MS-TCN (Chapter 4)	52.6	48.1	37.9	61.7	66.3
	MS-TCN++ (Chapter 4)	64.1	58.6	45.9	65.6	67.6
	BCN (*Wang et al.*, 2020)	68.7	65.5	55.0	66.2	70.4
	ASRF (*Ishikawa et al.*, 2021)	74.3	68.9	56.1	72.4	67.6
Timestamps	Seg. Model $\mathcal{M}$ + plateau (*Moltisanti et al.*, 2019)	65.5	59.1	43.2	65.9	63.5
	Ours	**70.5**	**63.6**	**47.4**	**69.9**	**64.1**
Transcripts	CDFL (*Li et al.*, 2019)	-	-	-	-	50.2
	MuCon (*Souri et al.*, 2021)	-	-	-	-	47.1
	D^{3}TW (*Chang et al.*, 2019)	-	-	-	-	45.7
	NN-Viterbi (*Richard et al.*, 2018b)	-	-	-	-	43.0
	TCFPN (*Ding and Xu*, 2018)	-	-	-	-	38.4
	HMM-RNN (*Richard et al.*, 2017)	-	-	-	-	33.3
	ECTC (*Huang et al.*, 2016)	-	-	-	-	27.7
Sets	SCT (*Fayyaz and Gall*, 2020)	-	-	-	-	30.4
	SCV (*Li and Todorovic*, 2020)	-	-	-	-	30.2
	Action Sets (*Richard et al.*, 2018a)	-	-	-	-	23.3

Table 5.8: Comparison with different levels of supervision on the Breakfast dataset.

weaker levels of supervision in the form of transcripts or sets with a large margin. Our approach provides a good compromise between annotation effort and performance, and further reduces the gap to fully supervised approaches.

Timestamp supervision has recently been studied for action localization in (*Ma et al.*, 2020). In their approach, they use the model confidence to sample foreground action frames and background frames for training. To compare with (*Ma et al.*, 2020), we use the same setup and the provided human annotations to train our model and report mean average precision at different overlapping thresholds. Table 5.10 shows the results on the GTEA and BEOID (*Damen et al.*, 2014) datasets. Our approach outperforms (*Ma et al.*, 2020) with a large margin of 5.4% average mAP on GTEA

Supervision	Method	F1@{10, 25, 50}			Edit	Acc
Full	MS-TCN (Chapter 4)	85.8	83.4	69.8	79.0	76.3
	MS-TCN++ (Chapter 4)	88.8	85.7	76.0	83.5	80.1
	BCN (*Wang et al.*, 2020)	88.5	87.1	77.3	84.4	79.8
	ASRF (*Ishikawa et al.*, 2021)	89.4	87.8	79.8	83.7	77.3
Timestamps	Seg. Model $\mathcal{M}$ + plateau (*Moltisanti et al.*, 2019)	74.8	68.0	43.6	**72.3**	52.9
	Ours	**78.9**	**73.0**	**55.4**	**72.3**	**66.4**

Table 5.9: Comparison with different levels of supervision on the GTEA dataset.

mAP@IoU	0.1	0.3	0.5	0.7	Avg
GTEA					
SF-Net (*Ma et al.*, 2020)	58.0	37.9	19.3	11.9	31.0
Ours	**60.2**	**44.7**	**28.8**	**12.2**	**36.4**
BEOID					
SF-Net (*Ma et al.*, 2020)	62.9	**40.6**	16.7	3.5	30.1
Ours	**71.5**	40.3	**20.3**	**5.5**	**34.4**

Table 5.10: Comparison with SF-Net (*Ma et al.*, 2020) for action localization with timestamp supervision on the GTEA and BEOID datasets.

and 4.3% on the BEOID dataset. In contrast to (*Ma et al.*, 2020) where only the frames with high confidence are used for training, our approach detects action changes and generates a target label for each frame in the training videos.

Finally, we compare our approach with the semi-supervised setup on the Breakfast dataset proposed in (*Kuehne et al.*, 2020). In this setup, the training videos are annotated with the transcript of actions and a fraction of the frames as well. Compared to the proposed timestamp supervision, this setup provides annotations for much more frames. Since the timestamps are randomly sampled from the video, there are sometimes multiple timestamps for one action and not all actions are annotated. As shown in Table 5.11, our approach outperforms (*Kuehne et al.*, 2020) with a large margin. While the approach of (*Kuehne et al.*, 2020) relies on an expensive Viterbi decoding during inference time, our approach directly predicts the frame-wise labels.

Fraction	Method	Acc
0.1	HMM-RNN (*Kuehne et al.*, 2020)	60.9
	Ours	**68.4**
0.01	HMM-RNN (*Kuehne et al.*, 2020)	58.8
	Ours	**67.4**

Table 5.11: Comparison with Kuehne *et al.* (*Kuehne et al.*, 2020) on the Breakfast dataset with semi-supervised setup.

Human Annotations vs Generated Annotations

In Table 5.12, we directly compare the human annotations from (*Ma et al.*, 2020) with simulated annotations on the GTEA dataset. The results show that the performance with random sampling is very close to real annotations.

mAP@IoU	0.1	0.3	0.5	0.7	Avg
Human annotations	**60.2**	44.7	**28.8**	**12.2**	**36.4**
Random sampling	59.7	**46.3**	26.0	10.4	35.6

Table 5.12: Human vs. generated annotations on the GTEA dataset.

Impact of Noise

In Table 5.11, the timestamps are randomly sampled from the videos. Thus, there are sometimes multiple timestamps for one action and not all actions are annotated. Table 5.13 shows the percentage of action segments with 0, 1, or >1 timestamps (TS). As shown in the table, our approach is robust to annotation errors.

Fraction	% actions with 0 TS	% with 1 TS	% with > 1 TS	Acc
0.1	3.5	3.4	93.1	68.4
0.01	27.5	21.6	50.9	67.4
0.0032	49.0	25.2	25.8	61.7
0.0032	0	100	0	64.1

Table 5.13: Percentage of actions with 0, 1, or >1 timestamps (TS) using the protocol of Table 11. The last row is the protocol of Table 8.

5.4 Summary

In this chapter, we proposed an approach to train a temporal action segmentation model using only timestamps annotations. Our approach combines the model predictions with the timestamps annotations for estimating where the action labels change. We further introduced a confidence loss that enforces monotonicity on the model confidence. The loss encourages high confidence values for all frames within an action segment and suppresses outlier frames. Results on four datasets show that models trained with timestamp supervision achieve comparable performance to the fully supervised setup. The proposed approach is model agnostic and can be applied to any segmentation model.

CHAPTER 6

Anticipating Temporal Occurrences of Activities

In the previous two chapters, we demonstrated the capabilities of deep learning methods in analyzing human actions in videos. Despite the success of such methods, this is not enough for intelligent agents that interact with humans. Such intelligent agents require reasoning beyond what has happened and predict what will happen in the future. Approaches addressing this task are, however, limited to anticipating only the immediate future. In this chapter, we therefore propose to extend the anticipation task and make long-term predictions over more than just a few seconds. To address this long-term anticipation task, we propose two methods to predict a considerably large number of future actions and their durations. Both, a CNN and an RNN are trained to learn future video labels based on previously seen content. We show that our methods generate accurate predictions of the future even for long videos with a huge amount of different actions and can even deal with noisy or erroneous input information. This chapter is based on (*Abu Farha et al.*, 2018).

Contents

6.1 Introduction

In the last years, we have seen a tremendous progress in the capabilities of computer systems to classify and segment activities in videos, *e.g.* (*Richard et al.*, 2017; *Lea et al.*, 2017; *Lei and Todorovic*, 2018). These systems, however, analyze the past or in the case of real-time systems the present

with a delay of a few milliseconds. For applications, where a moving system has to react or interact with humans, this is insufficient. For instance, collaborative robots that work closely with humans have to anticipate the activities of a human in the future. In contrast to humans that are very good in anticipating activities, developing methods that anticipate future activities from video data is very challenging and has just recently received an increase of interest.

Current works anticipate activities only for a very short time horizon of a few seconds. While early activity detection addresses the problem of inferring the class label of an action at the point when the activity starts or shortly thereafter (*Ryoo*, 2011; *Hoai and De la Torre*, 2014; *Ma et al.*, 2016; *Sadegh Aliakbarian et al.*, 2017), other works predict the class label of the action that will happen next (*Pei et al.*, 2013; *Jain et al.*, 2016; *Lan et al.*, 2014). In the work (*Mahmud et al.*, 2017), the starting time of the future activity is estimated as well.

In this chapter, we go beyond the recognition of an ongoing activity or the anticipation of the next activity. We address the problem of anticipating all activities that will be happening within a time horizon of up to 5 minutes. This includes the classes and order of the activities that will occur as well as when each activity will start and end.

To address this problem, we propose two novel approaches for this task. In both cases, we first infer the activities from the observed part of the video using an off-the-shelf action segmentation model. The first approach builds on a recurrent neural network (RNN) that predicts for a given sequence of inferred activities the remaining duration of the ongoing activity as well as the duration and class of the next activity. The anticipated activities are then fed back to the RNN in order to anticipate activities for a longer time horizon. The second approach builds on a convolutional neural network (CNN). To this end, we convert the temporal sequence of inferred activities in a matrix that encodes both the length and the action label information. The CNN then predicts a matrix that encodes the length and the action labels of the anticipated activities. In contrast to the RNN approach, the CNN approach anticipates all activities in one pass.

We have evaluated the two approaches on two challenging datasets that contain long sequences and large variations. Both approaches outperform by a large margin two baselines, a grammar based baseline and a nearest neighbor baseline. While the RNN and CNN perform similarly for a long time horizon of more than 40 seconds, the RNN performs better for shorter time horizons less than 20 seconds. Both approaches also outperform the method of *Vondrick et al.* (2016) that does not anticipate activities directly but visual representations of future frames, which can then be used to classify the activities.

6.2 Anticipating Activities

Our goal is to anticipate from an observed video what will happen next in the video for a given time horizon, which can take up to 5 minutes. We aim to predict for each frame in the future the label of the activity that will happen. More formally, let $x_{1:T} = (x_1, \ldots, x_T)$ be a video with T frames. Given the first t frames $x_{1:t}$, the task is to predict the actions happening from frame $t+1$ to T. That is, we aim to assign action labels $a_{t+1:T} = (a_{t+1}, \ldots, a_T)$ to each of the unobserved frames.

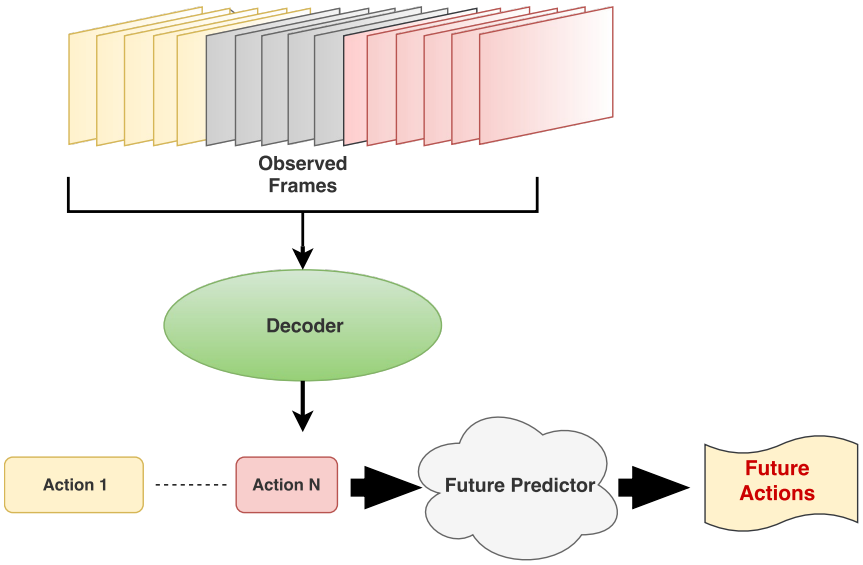


Figure 6.1: Proposed approach for future action prediction. From the observed frames, action labels are inferred by a decoder. The future predictor predicts from the inferred frame labels $a_{1:t}$ the labels $a_{t+1:T}$ that are yet to come.

6.2.1 Inferring Observed Activities

Instead of predicting the future actions $a_{t+1:T}$ directly from the video frames $x_{1:t}$, we first infer the actions $a_1, \ldots, a_t$ for the given frames $x_1, \ldots, x_t$ and then predict the future actions $a_{t+1}, \ldots, a_T$ from the inferred actions $a_{1:t}$ as it is illustrated in Figure 6.1. This has the advantage that we can separately study the impact of the network, which infers activities from observed video sequences, and the network that anticipates the future activities.

For inferring the activities $a_{1:t}$ from $x_{1:t}$, we experimented with two models. The first model is based on a hybrid RNN-HMM approach (*Richard et al.*, 2017). In contrast to (*Richard et al.*, 2017), which train the method in a weakly supervised setting, we train the model fully supervised since in our training set each video frame x_t is labeled with a class a_t. Whereas for the second model, we experimented with MS-TCN presented in Chapter 4 for inferring the activities $a_{1:t}$.

For predicting the activities $a_{t+1:T}$ from $a_{1:t}$, we investigate two architectures. The first architecture is based on a recurrent neural network (RNN) and will be described in Section 6.2.2. The second architecture is based on a convolutional neural network (CNN) and will be described in Section 6.2.3.

6.2.2 RNN-based Anticipation

We can interpret future action prediction as a recursive sequence prediction: As input sequence, the RNN obtains all observed segments and predicts the remainder of the last segment as well as the next segment. This is repeated until the desired amount of future frames is reached.

More precisely, for each observed segment, the RNN gets its class label in form of a 1-hot encoding and the corresponding segment length, which is normalized by the video length, as input. Sequentially forwarding all those segments, three output predictions are made: the remaining length of the last observed segment as well as a label and a length for the next segment. This prediction

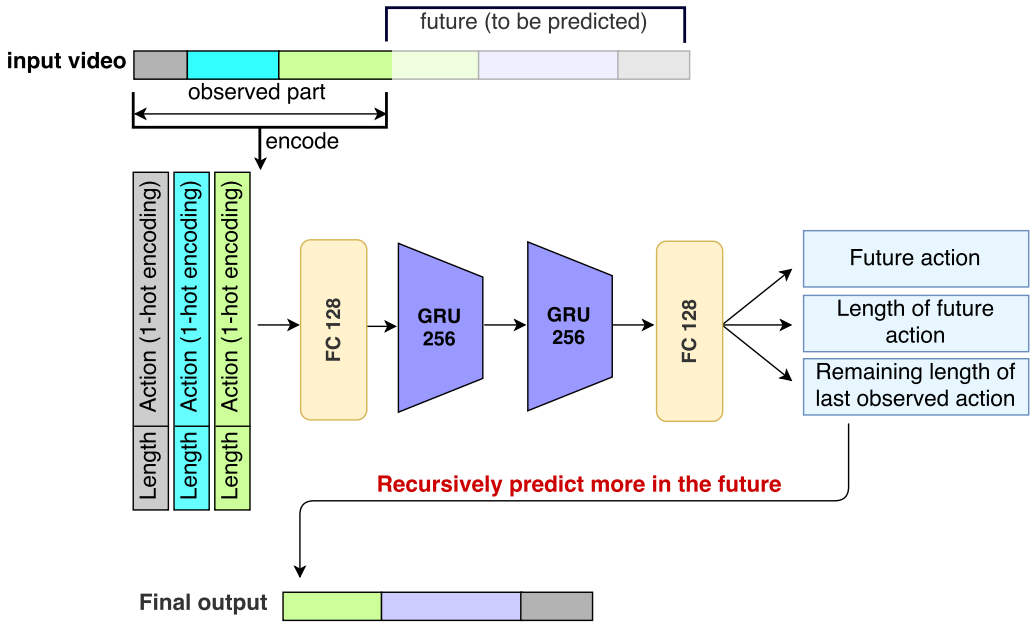


Figure 6.2: Architecture of the RNN system. The input is a sequence of *(length, 1-hot class encoding)*-tuples. The network predicts the remaining length of the last observed action and the label and length of the next action. Appending the predicted result to the original input, the next action segment can be predicted.

is concatenated with the observed segments to form a new input for the network. The new input is again forwarded through the network to produce the next prediction. The final result is obtained by repeatedly forwarding the previously generated prediction until the desired amount of frames is predicted, see Figure 6.2.

As RNN architecture, we use two stacked layers of 256 gated recurrent units and fully connected layers at the input and output. As output layer for both length predictions, remaining length of current action and length of next action, we use a rectified linear unit to ensure non-negative length outputs. The label prediction is done via a softmax layer as usual for classification tasks.

RNN Training. The training data generation for the RNN is illustrated in Figure 6.3. Given a ground-truth labeling of a training sequence with N action segments, $N-1$ training examples are generated out of it. For a segment $i < N$, a random split point is defined. Everything before this point is encoded as a sequence of i tuples containing the length of the observed segment and its label as 1-hot encoding. Each such sequence is an input training example for the network. For segment $i+1$, another random split point is defined. The values between the first and second split point define the target the network should predict: A triplet consisting of the remaining length of segment i (l_r), the length of the next action $i+1$ from its start up to the split point (l_n), and the label of the next action (a). By processing each training sequence like this, a large amount of input tuple sequences and target triplets is generated.

As loss for a single training example, we use

$$\mathcal{L} = -\log \hat{p}_a + (l_r - \hat{l}_r)^2 + (l_n - \hat{l}_n)^2, \tag{6.1}$$

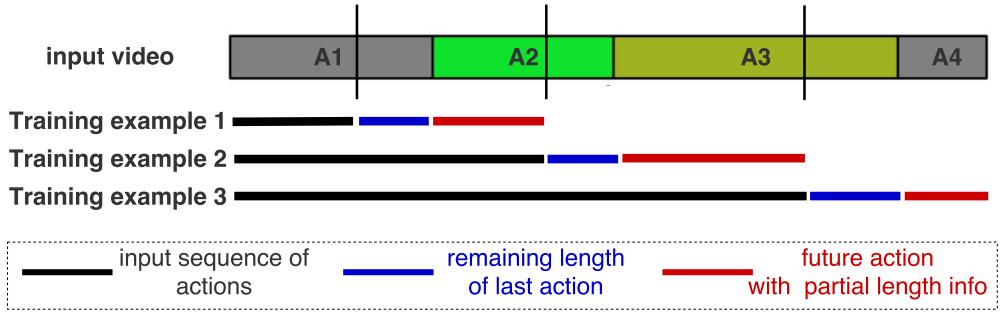


Figure 6.3: Training data is generated by cutting the ground-truth segmentations at random points and using the left part as input and the next action segment to the right of the cut as ground-truth for the prediction.

where $\hat{l}_r$ denotes the predicted remaining length of the current action, $\hat{l}_n$ denotes the predicted length of the next action, and $\hat{p}_a$ the predicted class probability of the next action. For training, we minimize the loss, which is summed over all training examples, by backpropagation through time.

6.2.3 CNN-based Anticipation

The CNN-based anticipation approach aims at predicting all actions directly in one single step, rather than relying on a recursive strategy such as the RNN. The given framewise labels $a_{1:t}$ are encoded in a matrix X with C columns and S rows. While the columns correspond to the C action classes, the rows correspond to action segments. The number of rows for an action segment of length l is given by $\lfloor \frac{l}{t} S \rfloor$ and, for each row s, $X_{sa} = 1$ for the label a of the corresponding action segment and zero otherwise. The matrix is filled in the order the actions occur as illustrated in Figure 6.4. We set S large enough such that each action segment covers at least one row.

The matrix X that encodes the observed labels $a_{1:t}$ is forwarded through a CNN which consists of two convolutional layers and two fully connected layers. The convolutional layers have 8 and 16 feature maps respectively and perform a 5×1 convolution followed by a rectified linear unit and max pooling. After the fully connected layers, the output layer is reshaped to a matrix Y, which has the same size as matrix X, and each row of Y is ℓ_2 normalized. We further apply a 1D Gaussian filter along each column for temporal smoothing. We will show in the experiments that the additional smoothing reduces the effect of spurious predictions of very short segments, see Figure 6.10. To convert Y back into a sequence of labels $a_{t+1:T}$, we compute for each row

$$\hat{a}_s = \arg\max_a Y_{sa} \tag{6.2}$$

and concatenate $\hat{a}_s$ where each row corresponds to $\lfloor \frac{T-t}{S} \rfloor$ frames.

CNN Training. Since the CNN approach predicts all actions directly while the RNN uses a recursive strategy, we also have to prepare the training data slightly differently. For each video in the training set, we generate 4 training examples by using the first 10%, 20%, 30%, and 50% of the video, respectively, as observation and the following 50% of the video as ground-truth for the prediction. For each training example, we convert the sequence of action labels $a_{1:t}$ into the matrix X

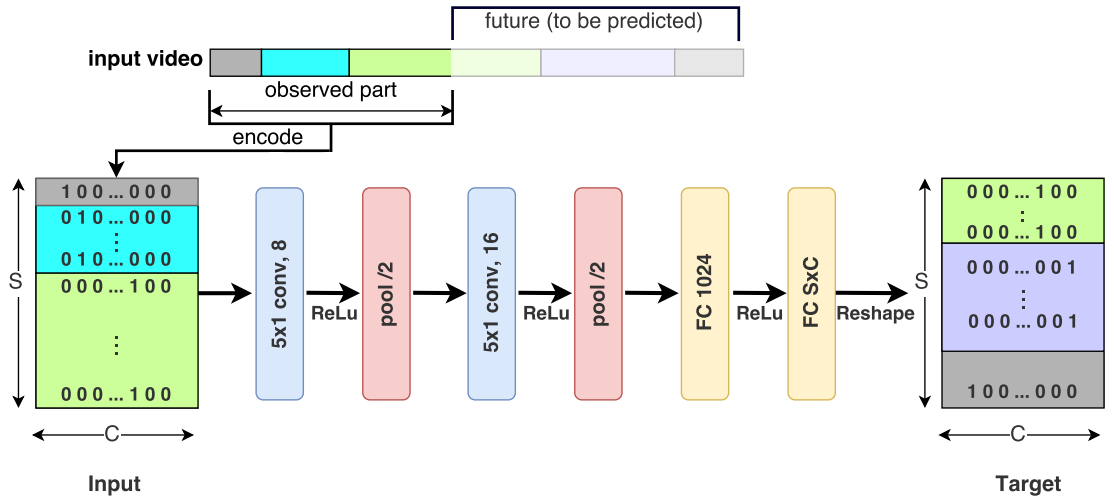


Figure 6.4: Architecture of the CNN-based anticipation approach. Both the input sequence and output sequence are converted into a matrix form where C denotes the number of classes and S corresponds to the number of video segments of a certain length. The binary values of the matrix indicate the label of each video segment.

and the labels of the following 50% of the video frames into the ground-truth matrix Y. To train the network, we use the squared error criterion over all output elements

$$\mathcal{L} = \frac{1}{SC} \sum_{s,a} (Y_{sa} - \hat{Y}_{sa})^2, \tag{6.3}$$

where $\hat{Y}$ is the prediction of the network. We found this loss in combination with the row-wise ℓ_2-normalization of the output more robust than a standard softmax output with cross-entropy loss which we attribute to the smoothing properties of the softmax function, see Section 6.3.5 for an evaluation.

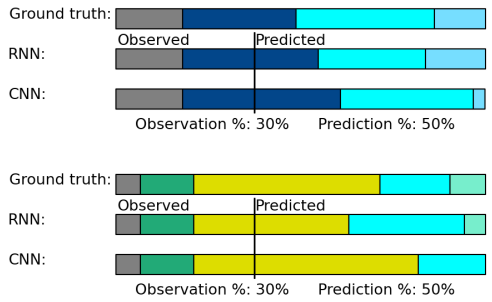
6.3 Experiments

6.3.1 Setup

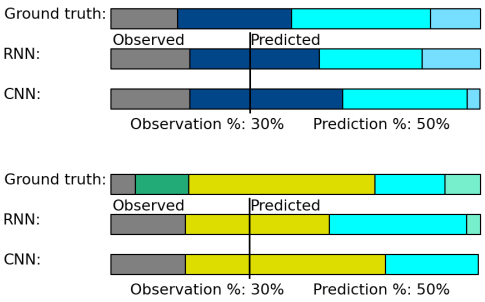
We conduct our experiments on two benchmark datasets for action recognition. The **Breakfast** dataset (*Kuehne et al.*, 2014) contains $1,712$ videos of 52 different actors making breakfast. Overall, there are 48 fine-grained action classes and about 6 action instances for each video. The average duration of the videos is 2.3 minutes. We use the four splits as proposed in (*Kuehne et al.*, 2014).

The **50Salads** dataset (*Stein and McKenna*, 2013) contains 50 videos with 17 fine-grained action classes. With an average length of 6.4 minutes, the videos are comparably long and contain 20 action instances per video on average. Following (*Stein and McKenna*, 2013), we use five-fold cross-validation.

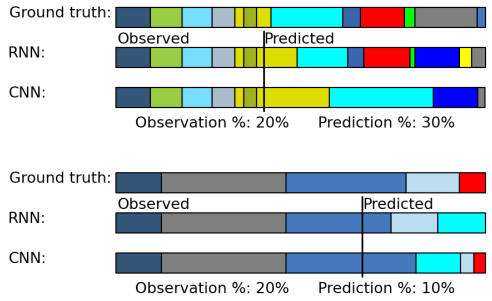
The longest video in both datasets is 10 minutes. More details about the datasets are available in Chapter 2. As evaluation metric, we report the accuracy of the predicted frames as mean over classes (MoC). Formal definitions for the evaluation metrics are discussed in Chapter 3.



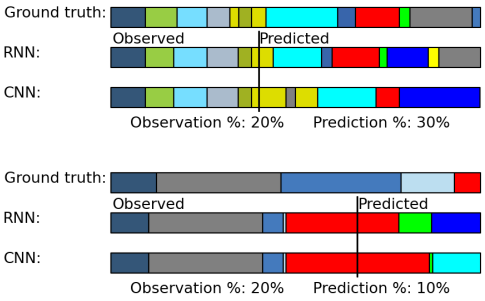
(a) Results on Breakfast with ground-truth observation



(b) Results on Breakfast without ground-truth observation



(c) Results on 50Salads with ground-truth observation



(d) Results on 50Salads without ground-truth observation

Figure 6.5: Qualitative results for the future action prediction task for both, RNN and CNN with and without ground-truth observations.

Video Representation. We evaluate our systems on two settings. The first is with ground-truth observations, *i.e.* the observed labels are the ground truth annotation. This setting allows for a clean analysis of the prediction capabilities of our systems. As a second setting, we consider the labels of the observed part of the videos to be obtained using the decoder from (*Richard et al.*, 2017) (or MS-TCN from Chapter 4). This way, already the observed labels can contain errors and prediction is much harder as previous errors are propagated into the future. In order to obtain the decoded labels from (*Richard et al.*, 2017), we compute improved dense trajectories over the observed frames $x_{1:t}$ and then reduce the dimensionality to 64 using PCA. After that, Fisher vectors are computed for each frame using a temporal window of size 20. For Fisher vectors computation, a Gaussian mixture model (GMM) with 64 Gaussians is built using $150,000$ random samples of the dense trajectory features. Power- and ℓ_2-normalization are applied, and at the end, the dimensionality is again reduced to 64 with PCA. For MS-TCN, we use I3D features as in Chapter 4.

Parameters. For the CNN approach, we set the number of rows S of the matrix X to 128 for the Breakfast dataset. This ensures that each action segment covers at least one row. Since the average video length for 50Salads is about four times larger than for Breakfast, we set S to 512 for 50Salads. As increasing S and therefore sampling at a finer temporal resolution did not significantly change the results, we stick to these values for the remainder of the chapter. For the post-processing, we use Gaussian smoothing with standard deviation $\sigma = 3$ for the Breakfast dataset and $\sigma = 13$ for 50Salads. For the RNN approach, the normalized length input is scaled by the average number of actions in the videos to ensure numerical stability.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
Grammar	48.92	40.33	36.24	31.46	52.66	42.15	38.44	33.09
Nearest-Neighbor	43.78	37.26	34.92	29.84	44.12	37.69	35.70	30.19
RNN model	**60.35**	**50.44**	**45.28**	**40.42**	**61.45**	**50.25**	44.90	**41.75**
CNN model	57.97	49.12	44.03	39.26	60.32	50.14	**45.18**	40.51
50Salads								
Grammar	28.69	21.65	18.32	10.37	26.71	14.59	11.69	09.25
Nearest-Neighbor	25.21	21.05	16.34	13.17	22.12	17.15	18.38	**14.71**
RNN model	**42.30**	**31.19**	**25.22**	**16.82**	**44.19**	**29.51**	19.96	10.38
CNN model	36.08	27.62	21.43	15.48	37.36	24.78	**20.78**	14.05

Table 6.1: Results for future action prediction with ground-truth observations. Numbers represent accuracy as mean over classes.

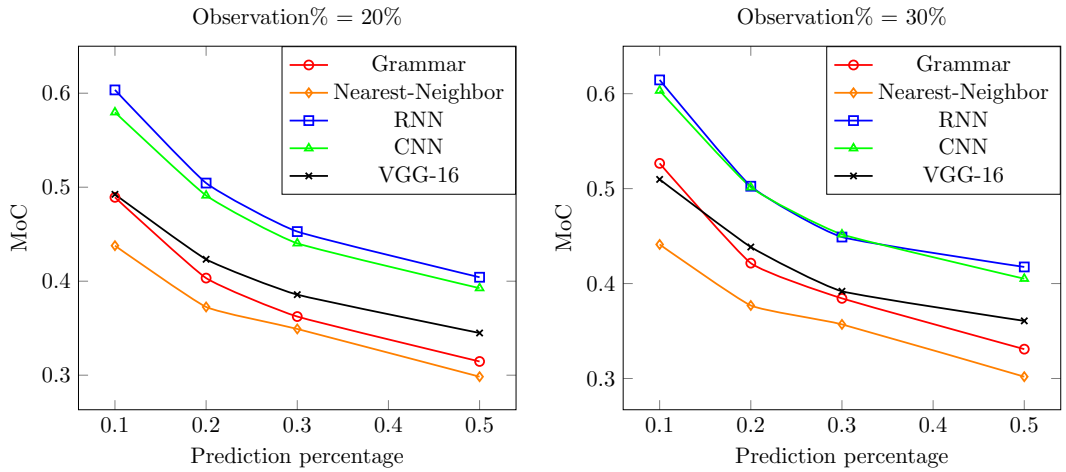
Baselines. We define two baselines for future action prediction to compare against our two proposed systems. The first baseline uses a grammar and the mean length of each action class. The mean length of each action class is estimated from the training data. The finite grammar generates all action sequences that have been observed during training. This is the same grammar as proposed in (*Richard et al.*, 2017). Given the observed actions $a_{1:t}$, either based on the ground truth or on the action decoder (*Richard et al.*, 2017), we randomly select an action sequence from the grammar that has $a_{1:t}$ as prefix. We then predict action labels $a_{t+1:T}$ such that the labels are consistent with the chosen action sequence of the grammar. Each action class from the chosen grammar path that has not been observed, *i.e.* that is to appear in the future, is added with its mean class length to the prediction until all required frames from $t+1$ to T are predicted.

As a second baseline, we use a nearest neighbor approach. We search the nearest neighbor in the training set using the frame-wise error of the observed part as a distance, and use the remaining part as future prediction.

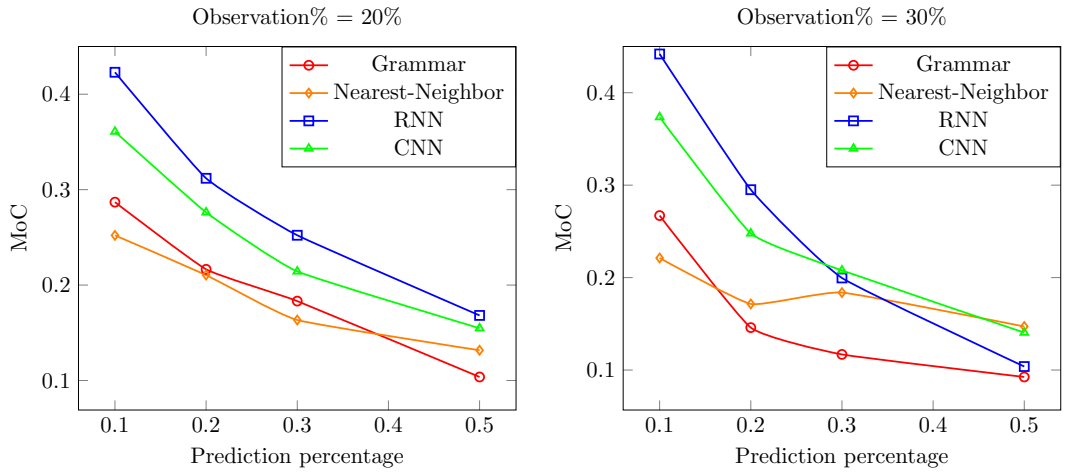
6.3.2 Prediction with Ground-Truth Observations

In order to provide a fair evaluation, we first assume that the observed segmentation $a_{1:t}$ of the video is perfect, *i.e.* that the observed labels do not contain any errors. While this is not the case in most realistic settings, it allows to get a clean evaluation how well the systems can predict the future. With noisy observations, the results are more delusive as errors in the observed part are propagated to the future.

In Table 6.1 and Figure 6.6, the results on the Breakfast dataset and 50Salads are shown. Both the RNN model and the CNN model show good performance compared to the baselines. Independent of the fraction of the video that has been observed, *i.e.* 20%, or 30%, however, the RNN outperforms the CNN in most cases. In general, the RNN is better for short term prediction, *i.e.* 10%, or 20%, while the CNN performs similarly or even sometimes better than the RNN for longer prediction. The reason



(a) Results on Breakfast



(b) Results on 50Salads

Figure 6.6: Results for future action prediction with ground-truth observations.

lies in the recursive structure of the RNN predictions: once a segment is predicted, it is appended to the observed part and used as input to predict the next sequence. Consequently, if the RNN outputs an erroneous prediction at some point, this error is likely to propagate through time. The CNN, on the contrary, uses the observed part of the video to predict all future actions directly, so errors are less likely to propagate from one segment to another. However, this slightly better performance of the CNN comes with the drawback of favouring long action segments over short ones. The behaviour can also be observed in the qualitative results in Figure 6.5 (a) and (c). For example in the case of (c), the CNN tends to miss small action segments, whereas the RNN seems to be more reliable.

6.3.3 Prediction without Ground-Truth Observations

In this section, we evaluate the performance of our proposed systems given noisy annotations. In contrast to the clean ground-truth observations from the previous section, we now assume that the

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
Grammar	16.60	14.95	13.47	13.42	21.10	18.18	17.46	16.30
Nearest-Neighbor	16.42	15.01	14.47	13.29	19.88	18.64	17.97	16.57
RNN model	**18.11**	**17.20**	**15.94**	**15.81**	21.64	20.02	**19.73**	**19.21**
CNN model	17.90	16.35	15.37	14.54	**22.44**	**20.12**	19.69	18.76
50Salads								
Grammar	24.73	22.34	**19.76**	12.74	29.65	19.18	15.17	13.14
Nearest-Neighbor	19.04	16.10	14.13	10.37	21.63	15.48	13.47	**13.90**
RNN model	**30.06**	**25.43**	18.74	**13.49**	**30.77**	17.19	14.79	09.77
CNN model	21.24	19.03	15.98	09.87	29.14	**20.14**	**17.46**	10.86

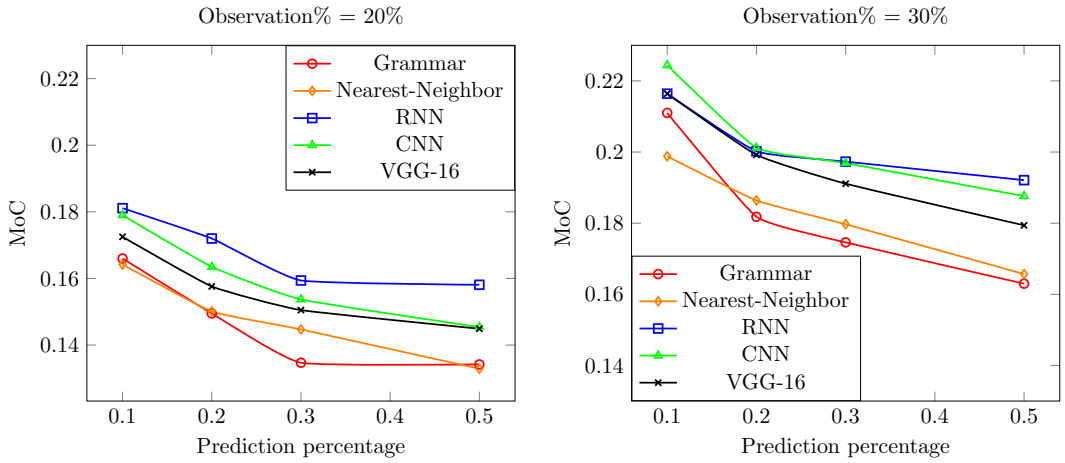
Table 6.2: Results for future action prediction without ground-truth observations. Numbers represent accuracy as mean over classes.

observed part of the video has been decoded using the system of (*Richard et al.*, 2017) and, thus, $a_{1:t}$ is not perfect anymore but is likely to contain errors. The mean over frames accuracy of (*Richard et al.*, 2017) when observing 20% of the video, for instance, is only 37% on Breakfast and 67% on 50Salads. While the accuracy when observing 30% of the video is 43% on Breakfast and 68% on 50Salads.

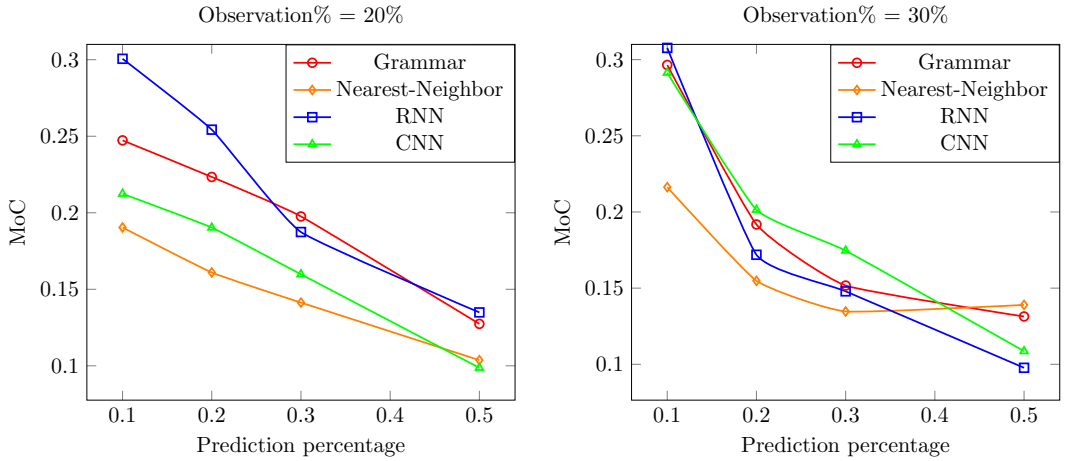
Compared to the amount of noise in the observed video labeling, the prediction results are still surprisingly stable, see Table 6.2 and Figure 6.7. While the drop in performance on the Breakfast dataset (Table 6.1 vs. Table 6.2) is comparably large, both systems still achieve a good performance compared to the baseline. On 50Salads, the overall loss of accuracy compared to the system with perfect observations is surprisingly small. This can on the one hand be attributed to the better performance of the decoder (*Richard et al.*, 2017). On the other hand, the inter-class dependencies on 50Salads are very strong, making it easier for both RNN and CNN to learn valid action sequences.

We would also like to put emphasis on the qualitative results for noisy observations, see Figure 6.5 (b) and (d). Particularly, the case of (d) is interesting: the decoder mistakenly predicted incorrect label for the last observed segment. For both models, RNN and CNN, this error propagates further to future segment predictions.

As the videos have different lengths, the proposed models might behave differently depending on the length of the videos. An evaluation on three categories of videos from the Breakfast dataset based on the prediction length is shown in Figure 6.8 for the case of observing 30% of the video and predicting the 50% that follows. While the models perform well on both short and long videos, they achieve a better performance for shorter videos. This is mainly because the duration of the predicted future is less for shorter videos, which makes the prediction task much easier compared to long video sequences. A similar behavior can also be observed when considering the number of actions that are predicted in the future. As shown in Table 6.3, the accuracy of both models drops as we predict more into the future.



(a) Results on Breakfast



(b) Results on 50Salads

Figure 6.7: Results for future action prediction without ground-truth observations.

	1st action	2nd action	3rd action
Grammar	22.32	13.25	11.17
Nearest-Neighbor	22.95	13.23	12.34
RNN	**26.43**	**17.73**	**17.92**
CNN	25.95	16.03	14.25

Table 6.3: Accuracy over next, 2nd action, and 3rd action for 30% observation and 50% prediction on the Breakfast dataset without ground-truth observations. The action is correctly detected if the IoU of the predicted action segment with the annotated action segment is $\geqslant 0.5$.

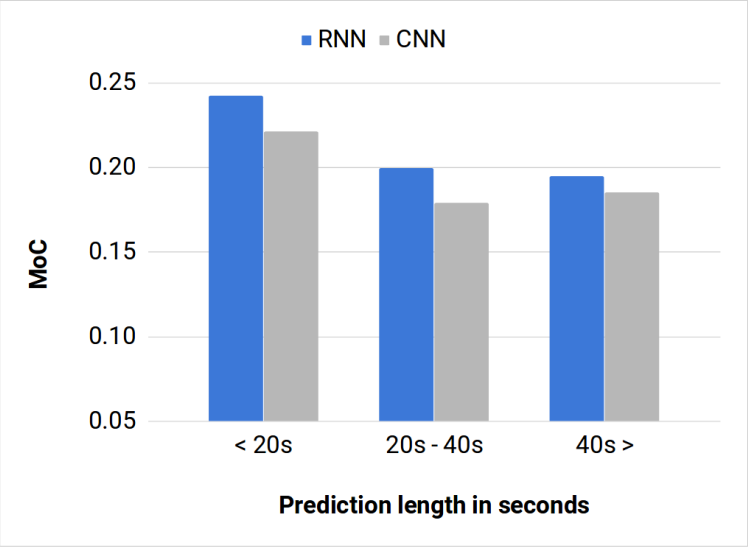


Figure 6.8: Performance of the models on videos with different lengths from the Breakfast dataset without ground-truth observations for the case of observing 30% of the videos and predicting the following 50%.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
RNN + RNN-HMM (*Richard et al.*, 2017)	18.11	17.20	15.94	15.81	21.64	20.02	19.73	19.21
RNN + MS-TCN (Chapter 4)	05.93	05.68	05.52	05.11	08.87	08.90	07.62	07.69
RNN + MS-TCN (Chapter 4)*	**23.56**	**20.74**	**19.52**	**18.37**	**26.12**	**23.55**	**21.47**	**20.99**
CNN + RNN-HMM (*Richard et al.*, 2017)	17.90	16.35	15.37	14.54	22.44	20.12	19.69	18.76
CNN + MS-TCN (Chapter 4)	09.85	09.17	09.06	08.87	17.59	17.13	16.13	14.42
CNN + MS-TCN (Chapter 4)*	**25.40**	**23.92**	**22.09**	**20.95**	**28.82**	**26.29**	**24.40**	**22.69**
50Salads								
RNN + RNN-HMM (*Richard et al.*, 2017)	30.06	25.43	18.74	13.49	**30.77**	17.19	14.79	09.77
RNN + MS-TCN (Chapter 4)	**32.31**	**25.51**	19.10	14.15	26.14	17.69	16.33	**12.97**
RNN + MS-TCN (Chapter 4)*	29.91	24.10	**20.48**	**15.22**	28.60	**20.09**	**17.55**	12.88
CNN + RNN-HMM (*Richard et al.*, 2017)	**21.24**	**19.03**	**15.98**	09.87	**29.14**	**20.14**	**17.46**	10.86
CNN + MS-TCN (Chapter 4)	16.02	14.68	12.09	**09.89**	19.23	14.68	13.18	**11.20**
CNN + MS-TCN (Chapter 4)*	15.62	14.48	11.64	9.71	17.16	12.66	11.11	10.55

Table 6.4: Impact of the recognition model on the anticipation performance. Numbers represent accuracy as mean over classes. * indicates that MS-TCN is trained with partial sequences that correspond to only the first $15 - 35\%$ of the videos.

6.3.4 Impact of the Recognition Model

In the previous section, we evaluated the proposed RNN and CNN models when the input frame-wise labels are obtained using the RNN-HMM (*Richard et al.*, 2017) decoder. In this section, we evaluate our methods when the inputs are inferred using the MS-TCN presented in Chapter 4. Since the observed part of the videos corresponds to only 20% or 30% of the video duration, we consider two variants of MS-TCN. The first variant is trained on the full extent of the videos and then used to infer the frame-wise labels for the observed part of the testing videos. Whereas the second variant is trained using only the first $15 - 35\%$ of the videos.

Table 6.4 shows the results on both 50Salads and the Breakfast dataset. While the RNN performs well using any of the recognition models on 50Salads, the CNN achieves slightly better performance when the RNN-HMM decoder is used. On the contrary, both models achieve significantly better performance using MS-TCN trained with partial videos on the Breakfast dataset. However, if the MS-TCN is trained on the full videos, the performance significantly drops. This is due to the difference between the training and inference setup (*i.e.* using full vs. partial videos as input). Training MS-TCN on the full videos results in falsely predicting labels at the end of the observations that usually occur at the end of the full videos. These errors are propagated to the anticipation models and severely affect the performance. Nonetheless, by using the same setup during both training and inference, the MS-TCN generates better predictions and the performance of the anticipation models is recovered. The mean over frames accuracy of the MS-TCN trained on full videos on the Breakfast dataset is only 23% and 32% for observing 20% and 30% of the videos respectively. If the MS-TCN is trained on partial sequences, then the mean over frames accuracy increases to 49% and 52% for observing 20% and 30% of the videos respectively. However, we noticed this behavior only on the

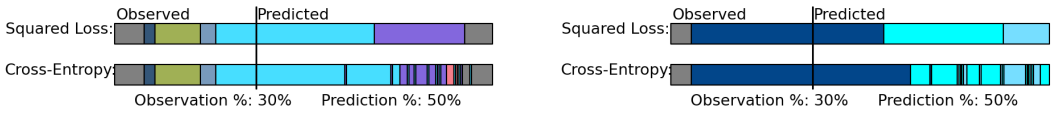


Figure 6.9: Results of the CNN model using the cross-entropy loss vs. the squared-error loss.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
CNN (cross-entropy)	51.77	42.80	39.40	36.25	54.99	48.38	43.24	38.58
CNN (squared loss)	**57.97**	**49.12**	**44.03**	**39.26**	**60.32**	**50.14**	**45.18**	**40.51**

Table 6.5: Comparing different loss functions for the CNN model on the Breakfast dataset with ground-truth observations. Numbers represent accuracy as mean over classes.

Breakfast dataset and training MS-TCN on the full videos of 50Salads achieves around 6.5% higher accuracy than training on partial videos. More discussion on the impact of the recognition model will be provided in Chapter 8, where we address this issue by proposing an end-to-end framework for long-term anticipation.

6.3.5 Analysis of the CNN Model

Model Architecture Compared to commonly used CNN architectures such as VGG-16 or ResNet, our model is comparably small. We stick to this simple architecture since we have very limited amount of training data, and complex models would easily overfit on the training set. A comparison of our architecture to a VGG-16 (*Simonyan and Zisserman*, 2015) on Breakfast with and without ground truth annotation is provided in Figure 6.6 (a) and Figure 6.7 (a), respectively. The performance of our architecture compared to VGG-16 is clearly better with an improvement up to 9% when using the ground truth annotation as observations. Note that our architecture already has around 6m parameters due to the fully connected layers at the end.

Loss Function Our choice of the loss function for the CNN based model is a squared loss preceded by an ℓ_2 normalization layer. However, for classification tasks, a softmax with cross-entropy loss is usually used. Since the future action prediction task can be viewed as special kind of classification, a softmax layer trained with the cross-entropy loss seems a more suitable choice on first glance. Yet, a comparison of both loss types shows a clear superiority of the squared loss combined with the ℓ_2 normalization layer, see Table 6.5. We attribute this to the smoothing properties of the softmax layer. Even large differences of the input values to a softmax layer lead to comparably small differences in the output probability distribution. This is a desirable effect in classification tasks. For our task, however, it leads to frequent changes of the maximizing label index, which corresponds to over-segmentations. The effect can also be observed in Figure 6.9. While strong over-segmentation can be observed for the softmax output, this effect nearly completely vanishes using the ℓ_2 normalization layer. Nevertheless, even with the ℓ_2 normalization layer, a small over-segmentation effect is still visible in some cases, which can be eliminated by the post-processing step as shown in Figure 6.10.

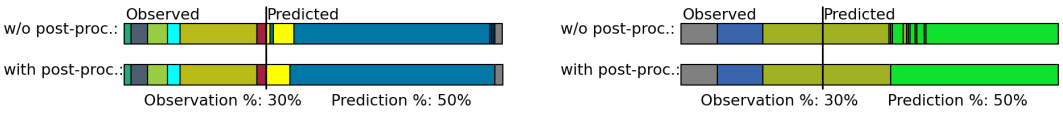


Figure 6.10: Results of the CNN model with and without post-processing.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
CNN (features)	12.78	11.62	11.21	10.27	17.72	16.87	15.48	14.09
CNN (w/o GT obs.)	**17.90**	**16.35**	**15.37**	**14.54**	**22.44**	**20.12**	**19.69**	**18.76**

Table 6.6: Results for future action prediction directly from features on the Breakfast dataset. Numbers represent accuracy as mean over classes.

6.3.6 Future Prediction Directly from Features

So far, we considered observations that are either frame-wise ground truth action labels, or those labels that are obtained by decoding the observed frame-wise features. In this section, we evaluate the performance of our models when applied directly on the observed frame-wise features and compare it to the two-step approach. We only use the CNN model for this evaluation. Originally, the input of the model is a matrix with C columns that correspond to the number of classes, and S rows that represent the temporal resolution. Each row is a 1-hot encoding that represents the action label of the corresponding frames in the observed sequence. When applying this model to features directly, C is equal to the dimensionality of the features, which is 64 in our case for the Fisher vectors features. S is kept at 128 as before. The observed video features are down-sampled or upsampled to have exactly 128 frames. We use the same training protocol that is used for the previous experiments, by considering the set $\{10\%, 20\%, 30\%, 50\%\}$ as observation percentages, and the target is always the 50% that follows immediately after the observations. Table 6.6 shows the results of the CNN model when applied on features directly compared to the two-step approach. As shown in the table, using the two-step approach outperforms the direct prediction from features by a large margin of up to $+5\%$. Predicting the future directly from features is a harder problem since the model has to recognize the observed actions and capture the relevant information to anticipate the future, while for the two-step approach these two tasks are decoupled. This allows to use a strong decoding model to recognize the observed actions, and restricting the future predictor to capture the context over action classes only instead of frame-wise features. A similar conclusion was reached by (*Garbade and Gall*, 2017) where semantic labels of the unseen parts are predicted in the spatial domain of an image. It has been shown that segmenting the observed part of the image first and then using the segmented image for prediction achieves better results than using the RGB image directly.

6.3.7 Comparison with the State-of-the-Art

To the best of our knowledge, long term future action prediction has not been addressed before. Most works focus on predicting the immediate future. *Vondrick et al.* (2016), for instance, train a model to predict AlexNet features of a frame one second in the future. Based on these predictions, an SVM is trained to determine the action label of the future frame.

	Breakfast	50Salads
Vondrick et al. (2016)	8.1	6.2
RNN model	**30.1**	**30.1**
CNN model	27	29.8

Table 6.7: Comparison with *Vondrick et al.* (2016): Accuracy of predicting future actions one second before they start is reported.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
RNN model	18.11	17.20	15.94	15.81	21.64	20.02	19.73	19.21
CNN model	17.90	16.35	15.37	14.54	22.44	20.12	19.69	18.76
UAAA (mode) (Chapter 7)	16.71	15.40	14.47	14.20	20.73	18.27	18.42	16.86
Time-Cond. (*Ke et al.*, 2019)	18.41	17.21	16.42	15.84	22.75	20.44	19.64	19.75
End2End (Chapter 8)	25.88	23.42	22.42	21.54	29.66	27.37	25.58	25.20
TAR (I3D) (*Sener et al.*, 2020)	24.20	21.10	20.00	18.10	30.40	26.30	23.80	21.20
TAR (I3D + Labels) (*Sener et al.*, 2020)	**37.10**	**31.80**	**30.10**	**27.10**	**39.80**	**34.20**	**31.90**	**27.80**
50Salads								
RNN model	30.06	25.43	18.74	13.49	30.77	17.19	14.79	09.77
CNN model	21.24	19.03	15.98	09.87	29.14	20.14	17.46	10.86
UAAA (mode) (Chapter 7)	24.86	22.37	19.88	12.82	29.10	20.50	15.28	12.31
Time-Cond. (*Ke et al.*, 2019)	32.51	27.61	21.26	15.99	35.12	27.05	22.05	15.59
End2End (Chapter 8)	34.76	28.41	21.82	15.25	34.39	23.70	18.95	15.89
TAR (IDT + Labels) (*Sener et al.*, 2020)	34.70	25.90	23.70	15.70	34.50	26.10	19.00	15.50
AGG (*Piergiovanni et al.*, 2020)	**40.40**	**33.70**	**25.40**	**20.90**	**40.70**	**40.10**	**26.40**	**19.20**

Table 6.8: Comparison with the state-of-the-art approaches extending our models. Numbers represent MoC accuracy.

Since *Vondrick et al.* (2016) train their future prediction model on $600h$ of videos from the web, which are not made publicly available, we use the recent I3D network trained on the Kinetics dataset to generate deep CNN features that have shown to generalize extremely well on several action recognition datasets (*Carreira and Zisserman*, 2017). We run the approach of *Vondrick et al.* (2016) on both, Breakfast and 50Salads, and compare their results to our model. To provide a fair comparison, our model is trained in a way that the input sequences always end one second before the next action segment starts. The results are shown in Table 6.7. Our approach outperforms the system of *Vondrick et al.* (2016) by a large margin. Note that for both datasets, predicting an action label only based on a single frame is particularly hard and the framewise action classifier achieves less than 10% accuracy. In contrast to *Vondrick et al.* (2016), our approaches make use of temporal context of the previously observed video content, which is crucial for reliable predictions.

Inspired by our methods, several researchers extended our approach and further improved the results. Table 6.8 provides a comparison with these approaches. *Ke et al.* (2019) proposed an approach that, instead of recursively predicting future activities, predicts a single future action segment by conditioning the model on a time variable. To predict all the future activities, several forward

passes with different time values are needed. *Sener et al.* (2020) proposed a multi-scale temporal aggregation framework for learning representations by relating recent observations to the long-range past. The learned representation are then used as input to an LSTM to generate the sequence of future activities. *Piergiovanni et al.* (2020) proposed an adversarial generative grammar model and applied it for both action anticipation and forecasting 3D human poses. Besides these methods, we will also extend our approach in the next chapter to predict multiple possible future sequences. Furthermore, we will introduce an end-to-end framework for long-term anticipation in Chapter 8.

6.4 Summary

We have introduced two efficient methods to predict future actions in videos, which is a task that has not been addressed before. While most existing prediction approaches focus on early anticipation of an already ongoing action or predict at most one action in the future, our methods have been the first to predict video content of up to several minutes length. Proposing two models to address the task, an RNN and a CNN, we obtain accurate predictions that scale well along different datasets and videos with varying lengths, varying quality of observed data, and huge variations in the possible future actions. Our approach has been a source of inspiration for several follow-up works.

CHAPTER 7

Uncertainty-Aware Anticipation of Activities

In the previous chapter, we proposed the long-term anticipation task and extended the prediction time horizon to several minutes into the future. However, with increasing the time horizon of the predicted activities, the future becomes more uncertain and models that generate a single prediction fail at capturing the different possible future activities. In this chapter, we address the uncertainty modelling for predicting long-term future activities. Both an action model and a length model are trained to model the probability distribution of the future activities. At test time, we sample from the predicted distributions multiple samples that correspond to the different possible sequences of future activities. Our model is evaluated on two datasets and shows a good performance in capturing the multi-modal nature of future activities without compromising the accuracy when predicting a single sequence of future activities. This chapter is based on (*Abu Farha and Gall*, 2019b).

Contents

7.1 Introduction

Anticipating future activities in video has become an active research topic in computer vision. While earlier approaches focused on early activity detection (*Ryoo*, 2011; *Hoai and De la Torre*, 2014; *Ma et al.*, 2016; *Sadegh Aliakbarian et al.*, 2017), recent models predict activities a few seconds in the future (*Lan et al.*, 2014; *Jain et al.*, 2016; *Vondrick et al.*, 2016; *Gao et al.*, 2017c). However, predicting the future activity label shortly before it starts is not sufficient for many applications. Robots, for instance, that interact with humans to accomplish industrial tasks or help in housework need to anticipate activities for a long time horizon. Such long-term prediction would enable these

Figure 7.1: The anticipation task. Given an observed part of a video, we want to predict the future activities that might follow in the future with their durations. The model outputs a collection of samples to represent the uncertainty in the future actions.

robots to plan ahead to complete their tasks efficiently. Moreover, anticipating the activities of other interacting humans improves human-robot interaction.

In the previous chapter, we extended the prediction horizon up to a few minutes into the future and predict both future activities and their durations as well. While our approach generates good predictions, it does not take the uncertainty of the future into consideration. For example, given a video snippet that shows a person taking a cup from the cupboard, we cannot be sure whether the future activity would be pour water or pour coffee. Approaches that predict a single output and do not model the uncertainty in the future actions would fail in such cases. On the contrary, approaches that are capable of predicting all the possible outputs are preferable. Figure 7.1 illustrates a case where the future activities have multiple modes and the model has to predict all these modes.

In this chapter, we introduce a framework that models the probability distribution of the future activities and use this distribution to generate several possible sequences of future activities at test time. To this end, we train an action model that predicts a probability distribution of the future action label, and a length model which predicts a probability distribution of the future action length. At test time, we sample from these models a future action segment represented by an action label and its length. To predict more in the future, we feed the predicted action segment to the model and predict the next one recursively. We evaluate our framework on two datasets with videos of varying length and many action segments: the Breakfast dataset (*Kuehne et al.*, 2014) and 50Salads (*Stein and McKenna*, 2013). Our framework outperforms a baseline that predicts the future activities using n-grams as an action model and a Gaussian to predict the action length. Furthermore, we are able to achieve results that are comparable with the state-of-the-art if we use the framework to predict a single sequence of future activities.

7.2 Anticipating Activities

Given an observed part of a video with t frames and the corresponding frame-wise labels $a_{1:t} = (a_1, \ldots, a_t)$, we want to predict all the action segments and their lengths that will occur in the future unseen part of that video. The observed video sequence can also be represented as n action segments $A_{1:n} = (A_1, \ldots, A_n)$ with their corresponding lengths $\ell_{1:n} = (\ell_1, \ldots, \ell_n)$. And the target are the segments $A_{n+1:N}$ with the corresponding segments lengths $\ell_{n+1:N}$, where N is the total number of action segments in the video. Furthermore, since the last observed action segment A_n might be partially observed and will continue in the future, we want to update our estimate of ℓ_n as well. Since for the same observed action segments $A_{1:n}$ there are more than one possible future action segments, we want to predict more than one output to capture the different modes in the future as shown in Figure 7.1. To this end, we propose a framework to model the uncertainty in the future activities, and

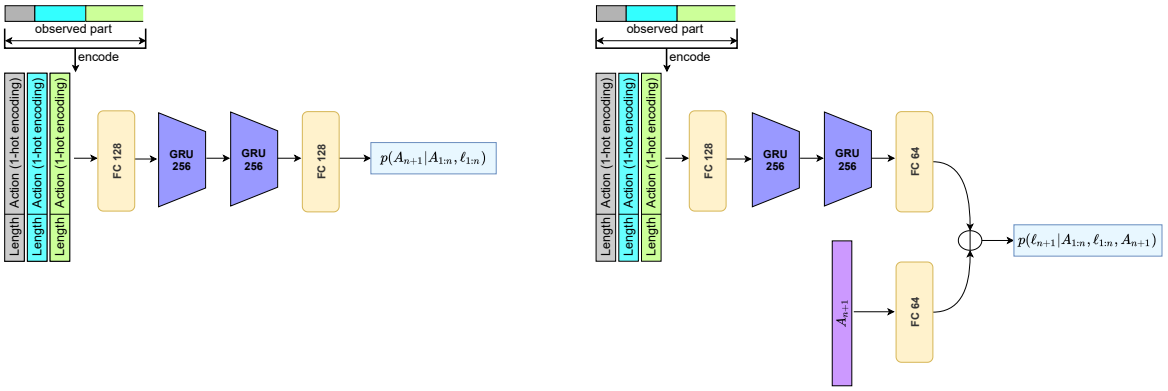


Figure 7.2: Our anticipation framework consists of: (a) The action model which predicts the probability distribution of the future action label. (b) The length model which predicts the probability distribution of the future action length.

then use this framework to generate samples of the future action segments. We start with the model description in Section 7.2.1, and then describe the prediction procedure in Section 7.2.2.

7.2.1 Model

We follow the two-step approach that we proposed in the previous chapter and infer the actions in the observed frames and then predict the future actions. For inferring the actions in the observed part, we also use the same RNN-HMM model (*Richard et al.*, 2017). Our goal now is to model the probability of the future actions and their lengths. This can be done using an autoregressive model that predicts the future action and its length, and then feeding the predicted output to the model again to predict the next one. Using such an autoregressive approach allows us to use the same model to predict actions for arbitrarily long time horizons. The probability distribution of the future action segment and its length can be factorized as follows

$$p(A_{n+1},\ \ell_{n+1}|A_{1:n},\ \ell_{1:n}) = p(A_{n+1}|A_{1:n},\ \ell_{1:n})\ \cdot\ p(\ell_{n+1}|A_{1:n},\ \ell_{1:n},\ A_{n+1}), \tag{7.1}$$

where the first factor $p(A_{n+1}|A_{1:n},\ \ell_{1:n})$ is an action model that describes the probability distribution of the future action label given the preceding action segments. Whereas the second factor $p(\ell_{n+1}|A_{1:n},\ \ell_{1:n},\ A_{n+1})$ is a length model that describes the probability distribution of the future action length given the preceding action segments and the future action label. In the following, we discuss the details of these models.

7.2.1.1 Action Model

The action model predicts a probability distribution of the future action given the sequence of observed action segments and their lengths. For this model we use an RNN-based model similar to

the one proposed in the previous chapter as shown in Figure 7.2 (a). Given the observed part of the video, each action segment is represented by a vector of a 1-hot encoding of the action label and the corresponding segment length. This sequence is passed through a fully connected layer, two layers of gated recurrent units (GRUs), and another fully connected layer. We use ReLU activations for the fully connected layers. For the output layer, we use another fully connected layer that predicts action scores for the future action. To get the probability distribution of the future action, we apply a softmax over the predicted scores

$$p(A_{n+1} = \hat{A}|A_{1:n},\ \ell_{1:n}) = \frac{e^{x_{\hat{A}}}}{\sum_{\tilde{A}} e^{x_{\tilde{A}}}}, \tag{7.2}$$

where $x_{\hat{A}}$ is the predicted action score for the class $\hat{A}$. To train this model, we use a cross entropy loss

$$\mathcal{L}_{action} = \frac{1}{M}\sum_{m} -log(y_{m,A}), \tag{7.3}$$

where $y_{m,A}$ is the predicted probability for the ground-truth future action label A in the $m\text{-}th$ training example.

7.2.1.2 Length Model

We model the probability distribution of the future action length with a Gaussian distribution, *i.e.*

$$p(\ell_{n+1}|A_{1:n},\ \ell_{1:n},\ A_{n+1}) = \mathcal{N}(\mu,\ \sigma^2). \tag{7.4}$$

To predict the mean length μ and the standard deviation σ of this distribution, we use a network with two branches as shown in Figure 7.2 (b). The first branch is the same as the RNN-based model used in the action model. It takes the sequence of the observed action segments as input and encodes them into a single vector representation. Whereas the second branch is a single fully connected layer that takes a 1-hot encoding of the future action A_{n+1} and encodes it in another single vector representation. These two encoded vectors are concatenated and passed through two fully connected output layers to predict the mean length μ and the standard deviation σ. To ensure that the standard deviation is positive we use exponential activations for the output layer. To train the length model, we minimize the negative log likelihood of the target length

$$\begin{aligned}\mathcal{L}_{length} &= \frac{1}{M}\sum_{m} -log\ p(\ell_{n+1} = \ell_m^{gt}|A_{1:n},\ \ell_{1:n},\ A_{n+1}),\\ &= \frac{1}{M}\sum_{m} 0.5\ log(2\pi)\ + log(\sigma_m)\ +\ \frac{(\ell_m^{gt}\ -\ \mu_m)^2}{2\sigma_m^2},\end{aligned} \tag{7.5}$$

where ℓ_m^{gt} is the ground-truth length of the future action, μ_m and σ_m are the predicted mean length and standard deviation for the $m\text{-}th$ training example. As the first term in (7.5) is constant, our final loss function for the length model can be reduced to

$$\mathcal{L}_{length} = \frac{1}{M}\sum_{m} log(\sigma_m)\ + \frac{(\ell_m^{gt}\ -\ \mu_m)^2}{2\sigma_m^2}. \tag{7.6}$$

The training examples for both the action and length models are generated based on the ground-truth segmentation of the training videos. Given a video with N action segments, we generate $N-1$ training examples. For a segment $i > 1$, the sequence of all the preceding segments is considered as input of the training example, where each segment is represented by a vector of a 1-hot encoding of the action label and the corresponding segment length. The action label of segment i defines the target for the action model and its length serves as a target for the length model.

7.2.2 Prediction

Given an observed part of a video with n action segments, we want to generate plausible sequences of future activities. Note that the observed part might end in a middle of an action segment and the last segment in the observations might be not fully observed. In the following we describe two strategies for predicting the sequence of future activities. The first strategy generates multiple sequences of activities by sampling from the predicted distributions of our approach. Whereas the second strategy is used to generate a single prediction that corresponds to the mode of the predicted distributions.

7.2.2.1 Prediction by Generating Samples

At test time we alternate between two steps: sampling a future action label from the action model

$$\hat{A}_{n+1} \sim p(A_{n+1}|A_{1:n},\ \ell_{1:n}), \tag{7.7}$$

and then we sample a length for the sampled future action using the length model

$$\hat{\ell}_{n+1} \sim p(\ell_{n+1}|A_{1:n},\ \ell_{1:n},\ \hat{A}_{n+1}). \tag{7.8}$$

We feed the predicted action segment recursively to the model until we predict the desired time horizon. As the last observed action segment might continue in the future, we start the prediction with updating the length of the last observed action segment. For this step, we sample a length for the last observed action segment based on the preceding segments and only update the length of that segment if the generated sample is greater than the observed length as follows

$$\hat{\ell}_n \sim p(\ell_n|A_{1:n-1},\ \ell_{1:n-1},\ A_n), \tag{7.9}$$

$$\ell_n^\star = \begin{cases} \hat{\ell}_n & : \hat{\ell}_n > \ell_n \\ \ell_n & : otherwise \end{cases}, \tag{7.10}$$

where ℓ_n is the observed length of the last observed action segment, and $\ell_n^\star$ is the predicted full length of that segment.

7.2.2.2 Prediction of the Mode

For predicting the mode, we also alternate between predicting the future action label and then predicting the length of that label. However, instead of sampling from the predicted action distribution, we choose the action label with the highest probability. For predicting the length, we use the predicted mean length from the length model instead of sampling from the predicted distribution.

7.2.3 Implementation Details

We implemented both models in PyTorch (*Paszke et al.*, 2017) and trained them using Adam optimizer (*Kingma and Ba*, 2015) with a learning rate of 0.001. The batch size is set to 32. We trained the action model for 60 epochs and the length model is trained for 30 epochs. Dropout is used after each layer with probability 0.5. We also apply standardization to the length of the action segments as follows

$$\ell = \frac{\ell - \overline{\ell}}{\sigma_\ell}, \tag{7.11}$$

where $\overline{\ell}$ and σ_ℓ are the mean length and standard deviation, respectively, computed from the lengths of all action segments in the training videos.

7.3 Experiments

Datasets. We evaluate the proposed model on two challenging datasets: the Breakfast dataset (*Kuehne et al.*, 2014) and 50Salads (*Stein and McKenna*, 2013).

The **Breakfast** dataset contains $1,712$ videos with roughly 3.6 million frames. Each video belongs to one out of ten breakfast related activities, such as make tea or pancakes. The video frames are annotated with fine-grained action labels like *pour water* or *take cup*. Overall, there are 48 different actions where each video contains 6 action instances on average. The videos were recorded by 52 actors in 18 different kitchens with varying view points. For evaluation, we use the standard 4 splits as proposed in (*Kuehne et al.*, 2014) and report the average.

The **50Salads** dataset contains 50 videos with roughly $600,000$ frames. On average, each video contains 20 action instances and is 6.4 minutes long. All the videos correspond to salad preparation activities and were performed by 25 actors. The video frames are annotated with 17 different fine-grained action labels like *cut tomato* or *peel cucumber*. For evaluation, we use five-fold cross-validation and report the average as in (*Stein and McKenna*, 2013).

Evaluation Metric. We evaluate our framework using two evaluation protocols. The first protocol is used in the previous chapter where we observe 20% or 30% of the video and predict the following 10%, 20%, 30% and 50% of that video. As a metric, we report the mean over classes (MoC) by evaluating the frame-wise accuracy of each action class and then averaging over the total number of ground-truth action classes. To evaluate multiple samples of future activities, the average frame-wise accuracy of each action class is used to compute the MoC. The average accuracy over samples has been used in other future prediction tasks like predicting future semantic segmentation (*Bhattacharyya et al.*, 2019) or predicting the next future action label (*Mehrasa et al.*, 2019). The second protocol is used in (*Mehrasa et al.*, 2019) where we predict only the next future action segment and report the accuracy of the predicted label.

Baseline. As a baseline we replace our action model with tri-grams, where the probability of the action is determined based on the preceding two action segments. For the length model, we assume the length of an action follows a Gaussian distribution with the mean and variance estimated from the training action segments of the corresponding action.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
bi-grams	45.11	35.03	30.94	25.69	45.78	36.29	31.48	26.12
tri-grams	45.95	37.59	34.13	29.52	48.09	40.30	35.86	30.60
four-grams	47.28	38.55	34.74	29.70	49.88	41.43	36.45	30.86
Ours	**50.39**	**41.71**	**37.79**	**32.78**	**51.25**	**42.94**	**38.33**	**33.07**
50Salads								
bi-grams	30.39	22.30	18.53	10.75	31.53	18.59	12.95	08.84
tri-grams	31.88	23.13	19.19	11.58	32.07	19.31	13.90	09.40
four-grams	30.42	22.53	18.31	11.25	30.79	18.89	13.09	09.58
Ours	**34.95**	**28.05**	**24.08**	**15.41**	**33.15**	**24.65**	**18.84**	**14.34**

Table 7.1: Results for anticipation with ground-truth observations. Numbers represent mean over classes (MoC) metric averaged over 25 samples.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
Baseline	49.54	40.38	36.83	32.71	52.25	42.95	38.92	33.80
Ours	**53.00**	**44.10**	**39.72**	**34.90**	**53.99**	**44.53**	**40.21**	**35.58**
50Salads								
Baseline	31.65	25.02	20.75	12.41	36.98	22.51	16.04	11.58
Ours	**38.10**	**30.10**	**26.33**	**16.51**	**40.00**	**29.27**	**23.17**	**15.48**

Table 7.2: Results for anticipation with ground-truth observations. Numbers represent mean over classes (MoC) metric of the predicted distribution mode.

7.3.1 Anticipation with Ground-Truth Observations

We start the evaluation by using the ground-truth annotations of the observed part. This setup allows us to isolate the effect of the action segmentation model which is used to infer the labels of the observed part (*i.e.* the RNN-HMM model). Table 7.1 shows the results of our model compared to n-grams baselines. For each example in the test set, we generate 25 samples and use the average accuracy of each class to compute the mean over classes metric (MoC). As shown in Table 7.1, our approach outperforms the baselines on both datasets and in all the test cases. This indicates that our model learns a better distribution of the future action segments represented by the generated samples. We also show the effect of using bi-grams or four-grams instead of the used tri-grams for the baseline. As shown in Table 7.1, the effect of using different n-grams model is small. While using more history gives a slight improvement on the Breakfast dataset, the tri-grams model performs better than four-grams on the 50Salads dataset, which contains much longer sequences. For the rest of the experiments, we stick with the tri-grams model for the proposed baseline.

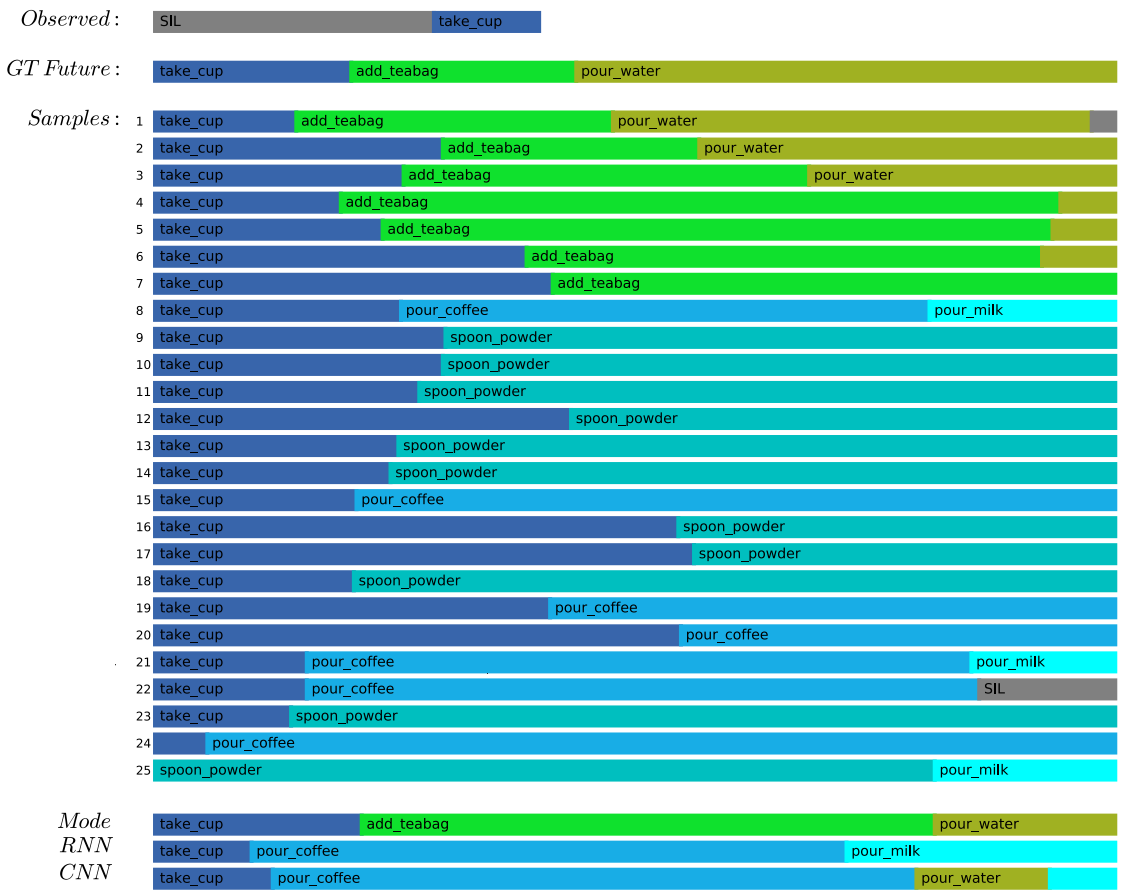


Figure 7.3: Qualitative result from the Breakfast dataset. This sequence corresponds to the activity of `making_tea`. We observe 20% of the video and predict the following 50%. Both the generated samples and the mode of the predicted distribution are shown. The samples are ranked based on the frame-wise accuracy of the predicted activities. We also show the results of the RNN and CNN models from the previous chapter.

We also report the accuracy of the mode of the predicted distribution. Instead of randomly drawing a sample from the predicted distribution, we predict the action label with the highest probability at each step. For the action length, we use the predicted mean length. Table 7.2 shows the results on both 50Salads and the Breakfast dataset. Our approach outperforms the baseline in this setup as well.

Figure 7.3 shows a qualitative result from the Breakfast dataset. Both the generated samples and the mode of the distribution are shown. We also show the results of the RNN and CNN models from the previous chapter. As illustrated in the figure, there are many possible action segments that might happen after the observed part (*SIL, take_cup*), and our model is able to generate samples that correspond to these different possibilities. In contrast, both the RNN and CNN from Chapter 6 generate only one possible future sequence of activities, which in this case does not correspond to the ground-truth.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
Baseline	15.39	13.65	12.93	11.90	**19.31**	16.56	15.76	13.90
Ours	**15.69**	**14.00**	**13.30**	**12.95**	19.14	**17.18**	**17.38**	**14.98**
50Salads								
Baseline	21.41	16.36	13.29	09.39	24.59	15.60	11.73	08.57
Ours	**23.56**	**19.48**	**18.01**	**12.78**	**28.04**	**17.95**	**14.77**	**12.06**

Table 7.3: Results for anticipation without ground-truth observations. Numbers represent mean over classes (MoC) metric averaged over 25 samples.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
Baseline	16.55	14.74	13.85	13.19	**20.80**	17.67	17.00	15.78
Ours	**16.71**	**15.40**	**14.47**	**14.20**	20.73	**18.27**	**18.42**	**16.86**
50Salads								
Baseline	21.74	17.43	14.96	10.34	28.06	18.70	14.60	09.75
Ours	**24.86**	**22.37**	**19.88**	**12.82**	**29.10**	**20.50**	**15.28**	**12.31**

Table 7.4: Results for anticipation without ground-truth observations. Numbers represent mean over classes (MoC) metric of the predicted distribution mode.

7.3.2 Anticipation without Ground-Truth Observations

In this section, we evaluate our approach without relying on the ground-truth annotations of the observed part. *I.e.* we infer the labels of the observed part of the video with the RNN-HMM model (*Richard et al.*, 2017), and then use our approach to predict the future activities. Table 7.3 reports the results of the generated samples, represented by the mean over classes averaged over 25 samples. Whereas Table 7.4 shows the results of the distribution mode. Our approach outperforms the baseline in both cases, which emphasizes the robustness of our approach to noisy input. Nevertheless, the baseline performs slightly better than our approach for the case of observing 30% and predicting the next 10% of the videos on the Breakfast dataset. This case corresponds to a short-term prediction where in most cases the future activities consist of only one action segment. Whereas our approach is better for longer time horizons.

7.3.3 Effect of the Number of Samples

To evaluate the generated samples, we report the mean over classes averaged over 25 samples. Table 7.5 shows the effect of using different numbers of samples. In each case, the average and standard deviation of 5 runs are reported. As shown in Table 7.5, the impact of the number of samples is small. While the average MoC remains in the same range, the standard deviation decreases as increasing the number of samples.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
5 samples	15.47 ± 00.14	14.02 ± 00.07	13.49 ± 00.20	13.10 ± 00.18	19.07 ± 00.17	17.03 ± 00.16	17.38 ± 00.36	15.08 ± 00.20
10 samples	15.57 ± 00.10	14.07 ± 00.12	13.47 ± 00.17	13.02 ± 00.13	18.97 ± 00.15	17.08 ± 00.12	17.07 ± 00.10	15.02 ± 00.10
25 samples	15.57 ± 00.07	14.06 ± 00.08	13.42 ± 00.07	13.10 ± 00.06	19.04 ± 00.06	17.12 ± 00.08	17.18 ± 00.11	15.09 ± 00.06
50 samples	15.56 ± 00.11	14.00 ± 00.06	13.48 ± 00.06	12.98 ± 00.03	19.06 ± 00.11	17.12 ± 00.05	17.22 ± 00.03	15.07 ± 00.06
50Salads								
5 samples	24.04 ± 02.00	20.02 ± 00.71	16.93 ± 00.77	12.47 ± 00.78	26.39 ± 01.08	17.98 ± 01.39	14.53 ± 01.18	11.99 ± 00.44
10 samples	23.04 ± 01.18	20.37 ± 00.45	17.39 ± 00.54	12.64 ± 00.54	26.99 ± 00.80	18.07 ± 00.75	14.68 ± 00.95	12.59 ± 00.40
25 samples	23.16 ± 00.68	20.03 ± 00.40	17.30 ± 00.48	12.71 ± 00.13	26.58 ± 00.42	18.22 ± 00.62	14.83 ± 00.60	12.20 ± 00.29
50 samples	23.08 ± 00.32	19.98 ± 00.10	17.54 ± 00.14	12.78 ± 00.13	26.64 ± 00.30	18.26 ± 00.36	14.59 ± 00.28	12.25 ± 00.37

Table 7.5: Effect of the number of samples. Numbers represent mean over classes (MoC) metric averaged over the samples. In each case, the average and standard deviation of 5 runs are reported.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
RNN model (Chapter 6)	60.35	50.44	45.28	40.42	61.45	50.25	44.90	41.75
CNN model (Chapter 6)	57.97	49.12	44.03	39.26	60.32	50.14	45.18	40.51
Time-Cond. (*Ke et al.*, 2019)	64.46	56.27	50.15	43.99	65.95	55.94	49.14	44.23
Ours (Mode)	53.00	44.10	39.72	34.90	53.99	44.53	40.21	35.58
Ours (Top-1)	78.84	72.84	66.29	63.45	82.00	72.83	69.13	62.39
50Salads								
RNN model (Chapter 6)	42.30	31.19	25.22	16.82	44.19	29.51	19.96	10.38
CNN model (Chapter 6)	36.08	27.62	21.43	15.48	37.36	24.78	20.78	14.05
Time-Cond. (*Ke et al.*, 2019)	45.12	33.23	27.59	17.27	46.40	34.80	25.24	13.84
Ours (Mode)	38.10	30.10	26.33	16.51	40.00	29.27	23.17	15.48
Ours (Top-1)	74.89	58.75	46.07	35.71	67.39	52.37	46.73	36.64

Table 7.6: Comparison with the state-of-the-art using ground-truth observations. Numbers represent mean over classes (MoC).

7.3.4 Comparison with the State-of-the-Art

In this section, we compare our approach with the state-of-the-art methods for anticipating activities. Since the state-of-the-art methods do not model uncertainty and predict just a single sequence of future activities, we report the accuracy of the mode of the predicted distribution of our approach. Table 7.6 shows the results with the ground-truth observations, and Table 7.7 shows the results without the ground-truth observations. While the mode of the distribution from our approach only outperforms the CNN model (Chapter 6) on the 50Salads dataset, it shows a lower accuracy compared to the RNN model and (*Ke et al.*, 2019). This is expected since these approaches were trained to predict only a single sequence of future activities while our approach is trained to predict multiple sequences. We therefore also report the top-1 MoC. The results show that our model captures the future activities better.

We also compare our approach with (*Mehrasa et al.*, 2019) in predicting the next action segment given all the previous segments. The accuracy of predicting the label of the future action segment is reported in Table 7.8. Note that in this comparison we use the ground-truth annotations of the videos as in (*Mehrasa et al.*, 2019). Our approach achieves a lower accuracy. However, our approach is designed for long-term prediction as we have already observed in Section 7.3.2 and this comparison considers short-term prediction only. Moreover, the approach of *Mehrasa et al.* (2019) is very expensive, making it infeasible for long-term predictions.

Finally, we compare with the follow-up work of *Zhao and Wildes* (2020). They proposed a conditional generative adversarial network based on LSTMs. To train the model, they used a combination of an adversarial loss and a normalized distance regularizer that encourages realistic and diverse predictions. Table 7.9 compares the results of *Zhao and Wildes* (2020) with our approach. While their approach outperforms ours on the average MoC metric, our approach is still competitive when the top-1 sample is considered.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
RNN model (Chapter 6)	18.11	17.20	15.94	15.81	21.64	20.02	19.73	19.21
CNN model (Chapter 6)	17.90	16.35	15.37	14.54	22.44	20.12	19.69	18.76
Time-Cond. (*Ke et al.*, 2019)	18.41	17.21	16.42	15.84	22.75	20.44	19.64	19.75
Ours (Mode)	16.71	15.40	14.47	14.20	20.73	18.27	18.42	16.86
Ours (Top-1)	28.89	28.43	27.61	28.04	32.38	31.60	32.83	30.79
50Salads								
RNN model (Chapter 6)	30.06	25.43	18.74	13.49	30.77	17.19	14.79	09.77
CNN model (Chapter 6)	21.24	19.03	15.98	09.87	29.14	20.14	17.46	10.86
Time-Cond. (*Ke et al.*, 2019)	32.51	27.61	21.26	15.99	35.12	27.05	22.05	15.59
Ours (Mode)	24.86	22.37	19.88	12.82	29.10	20.50	15.28	12.31
Ours (Top-1)	53.53	42.99	40.50	33.70	56.43	42.82	35.80	30.22

Table 7.7: Comparison with the state-of-the-art without ground-truth observations. Numbers represent mean over classes (MoC).

Model	Accuracy
APP-VAE (*Mehrasa et al.*, 2019)	**62.2**
Ours	57.8

Table 7.8: Comparison with (*Mehrasa et al.*, 2019): Accuracy of predicting the label of the next action segment.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
Ours (AVG)	50.39	41.71	37.79	32.78	51.25	42.94	38.33	33.07
Ours (Top-1)	78.84	72.84	66.29	63.45	82.00	72.83	69.13	62.39
Zhao and Wildes (2020) (AVG)	72.22	62.40	56.22	45.95	74.14	71.32	65.30	52.38
Zhao and Wildes (2020) (Top-1)	82.08	70.59	68.51	64.06	83.36	76.85	72.13	64.06
50Salads								
Ours (AVG)	34.95	28.05	24.08	15.41	33.15	24.65	18.84	14.34
Ours (Top-1)	74.89	58.75	46.07	35.71	67.39	52.37	46.73	36.64
Zhao and Wildes (2020) (AVG)	46.63	35.62	31.91	21.37	46.13	36.37	33.10	19.45
Zhao and Wildes (2020) (Top-1)	51.50	38.45	36.06	27.62	50.79	47.54	37.83	29.08

Table 7.9: Comparison with the follow-up work of *Zhao and Wildes* (2020) using ground-truth observations. Numbers represent mean over classes (MoC).

7.4 Summary

We presented a framework for modelling the uncertainty of future activities. Both an action model and a length model are trained to predict a probability distribution over the future action segments. At test time, we used the predicted distribution to generate many samples. Our framework is able to generate a diverse set of samples that correspond to the different plausible future activities. While our approach achieves comparable results to the state-of-the-art for short-term prediction, our approach is in particular useful for long-term prediction since for such scenarios the uncertainty in the future activities increases.

CHAPTER 8

Long-Term Anticipation of Activities with Cycle Consistency

In the last two chapters, we proposed several approaches to extend the prediction horizon up to several minutes into the future and anticipate a sequence of future activities with their durations. However, these approaches decouple the semantic interpretation of the observed sequence from the anticipation task. In this chapter, we propose a framework for anticipating future activities directly from the features of the observed frames and train it in an end-to-end fashion. Furthermore, we introduce a cycle consistency loss over time by predicting the past activities given the predicted future. Our framework achieved state-of-the-art results on two datasets when the work has been published: the Breakfast dataset and 50Salads. This chapter is based on (*Abu Farha et al.*, 2020).

Contents

8.1 Introduction

Long-term anticipation of activities beyond just a few seconds is a task with many practical application. In Chapter 6, we proposed two models that are able to predict activities several minutes into the future. These models inspired several researchers to address the long-term anticipation task and further improve the performance (*Gammulle et al.*, 2019a; *Ke et al.*, 2019). While these approaches successfully predict future activities and their duration, they decouple the semantic interpretation of the observed sequence from the anticipation task using a two-step approach.

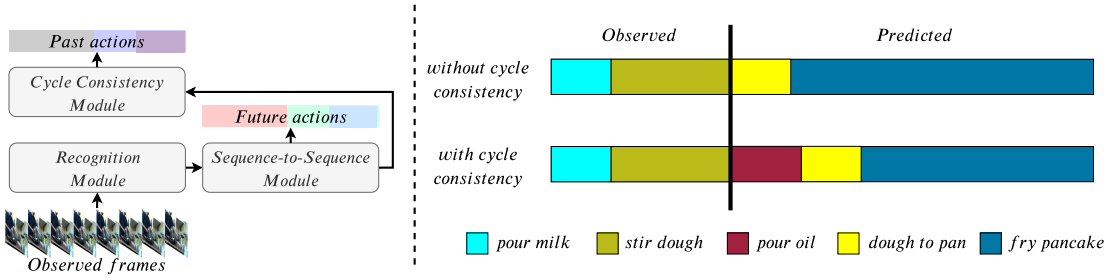


Figure 8.1: (Left) Overview of the proposed approach, which is trained end-to-end and includes a cycle consistency module. (Right) Effect of the cycle consistency module. Without the cycle consistency, the network anticipates actions that are plausible based on the previous actions. However, in some cases an essential action is missing. In this example *pour oil*. By using the cycle consistency, we enforce the network to verify if all required actions have been done before. For the action *fry pancake*, *pouring oil* into the pan is required and the cycle consistency resolves this issue.

Separating the understanding of the past and the anticipation of future has several disadvantages. First, the model is not trained end-to-end, which means that the approach for temporal action segmentation is not optimized for the anticipation task. Second, if there are any mistakes in the temporal action segmentation, these mistakes will be propagated and affect the anticipated activities. Finally, the action labels do not represent all information in the observed video that is relevant for anticipating the future. In contrast to these approaches, we propose a sequence-to-sequence model that directly maps the sequence of observed frames to a sequence of future activities and their duration. We then cast the understanding of the past as an auxiliary task by proposing a recognition module, which consists of a temporal convolutional neural network and a recognition loss, that is combined with the encoder.

Furthermore, as we humans can reason about the past given the future, previous works only aim to predict the future, and it is intuitive that forcing the network to predict the past as well is helpful. To this end, we propose a cycle consistency module that predicts the past activities given the predicted future. This module verifies if, for the predicted future actions, all required actions have been done before as illustrated in Figure 8.1. In this example, the actions *pour dough to pan* and *fry pancake* are plausible given the previous actions, but the action *pour oil* has been missed. The cycle consistency module then predicts from the anticipated actions, the observed actions. Since *pour dough to pan* and *fry pancake* are the inputs for the cycle consistency module, it will predict all the required preceding actions such as *pour milk*, *stir dough* and *pour oil*. However, as *pour oil* is not part of the observations, the cycle consistency module will have a high error, which steers the network to predict *pour oil* in the future actions.

Our contribution is thus three folded: First, we propose an end-to-end model for anticipating a sequence of future activities and their durations. Second, we show that the proposed recognition module improves the sequence-to-sequence model. Third, we propose a cycle consistency module that verifies the predictions.

We evaluate our model on two datasets with untrimmed videos containing many action segments: the Breakfast dataset (*Kuehne et al.*, 2014) and 50Salads (*Stein and McKenna*, 2013). Our model is able to predict the future activities and their duration accurately achieving superior results compared to the state-of-the-art methods.

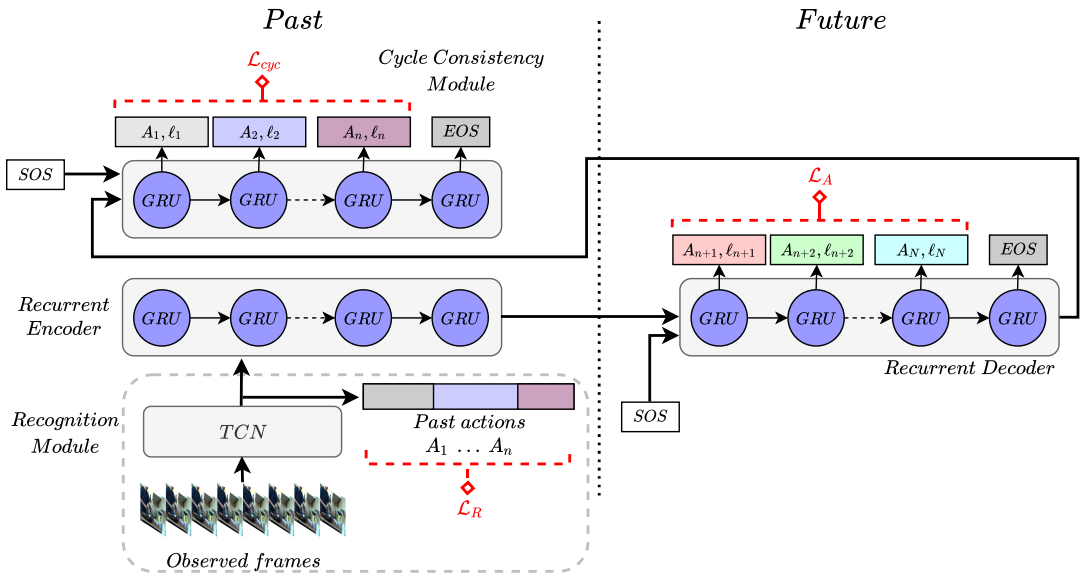


Figure 8.2: Overview of the anticipation framework. The observed frames are passed through a TCN-based recognition module which produces discriminative features for the sequence-to-sequence model. The sequence-to-sequence model predicts the future activities with their duration. In addition, we enforce cycle consistency over time by predicting the past activities given the predicted future.

8.2 The Anticipation Framework

Given a partially observed video with many activities, we want to predict all the activities that will be happening in the remainder of that video with their corresponding duration. Assuming that the observed part consists of t_o frames $x_{1:t_o} = (x_1, \ldots, x_{t_o})$ corresponding to n activities $A_{1:n} = (A_1, \ldots, A_n)$, our goal is to predict the future activities $A_{n+1:N} = (A_{n+1}, \ldots, A_N)$ and their corresponding duration $\ell_{n+1:N} = (\ell_{n+1}, \ldots, \ell_N)$, where N is the total number of activities in that video. In contrast to the previous approaches that use only the action labels of the observed frames for anticipating the future, we propose to anticipate the future activities directly from the observed frames. First, we propose a sequence-to-sequence model that maps the sequence of features from the observations to a sequence of future activities and their duration. Then, we introduce a cycle consistency module that predicts the past activities given the predicted future. The motivation of this module is to force the sequence-to-sequence module to encode all the relevant information in the observed frames and verify if the predictions are plausible. Finally, we extend the sequence-to-sequence model with a recognition module that generates discriminative features that capture the relevant information for anticipating the future. An overview of the proposed model is illustrated in Figure 8.2. Our framework is hence trained using an anticipation loss, a recognition loss, and a cycle consistency loss. In the following sections, we describe in detail these modules.

8.2.1 Sequence-to-Sequence Model

The sequence-to-sequence model maps the sequence of observed frames to a sequence of future activities and their duration. For this model, we use a recurrent encoder-decoder architecture based

on gated recurrent units (GRUs).

Recurrent Encoder. The purpose of the recurrent encoder is to encode the observed frames in a single vector which will be used to decode the future activities. Given the input features x, the recurrent encoder passes these features through a single layer with a gated recurrent unit (GRU)

$$h_t^e = GRU(x_t,\ h_{t-1}^e), \tag{8.1}$$

where x_t is the input feature for frame t, and h_{t-1}^e is the hidden state at the previous time step. The hidden state at the last time step $h_{t_o}^e$ encodes the observed frames and will be used as an initial state for the decoder.

Recurrent Decoder. Given the output of the encoder $h_{t_o}^e$, the recurrent decoder predicts the future activities and their relative duration. The decoder also consists of a single layer with a gated recurrent unit (GRU). The hidden state at each step is updated using the GRU update rules

$$h_m^d = GRU(A_{m-1},\ h_{m-1}^d), \tag{8.2}$$

where the input at each time step A_{m-1} is the predicted activity label for the previous step. At training time, the ground-truth label is used as input. Whereas the predicted label is used during inference time. For the first time step, a special start-of-sequence (SOS) symbol is used as input. Given the hidden state h_m^d at each time step, the future activity label A_m and its relative duration ℓ_m are predicted using a fully connected layer, *i.e.*

$$\tilde{A}_m = W_A h_m^d + b_A, \tag{8.3}$$

$$\hat{\ell}_m = W_\ell h_m^d + b_\ell, \tag{8.4}$$

where $\tilde{A}_m$ is the predicted logits for the future activity, and $\hat{\ell}_m$ is the predicted duration in log space. To get the relative duration, we apply softmax over the time steps

$$\tilde{\ell}_m = \frac{e^{\hat{\ell}_m}}{\sum_k e^{\hat{\ell}_k}}. \tag{8.5}$$

The decoder keeps predicting future activity labels and the corresponding duration until a special end-of-sequence (EOS) symbol is predicted. As a loss function, we use a combination of a cross entropy loss for the activity label and a mean squared error (MSE) for the predicted relative duration

$$\mathcal{L}_A = \mathcal{L}_{CE} + \mathcal{L}_{MSE}, \tag{8.6}$$

$$\mathcal{L}_{CE} = \frac{1}{N-n} \sum_{m=n+1}^{N} -log(\tilde{a}_{m,A}), \tag{8.7}$$

$$\mathcal{L}_{MSE} = \frac{1}{N-n} \sum_{m=n+1}^{N} (\tilde{\ell}_m - \ell_m)^2, \tag{8.8}$$

where $\mathcal{L}_A$ is the anticipation loss, $\tilde{a}_{m,A}$ is the the predicted probability for the ground truth activity label at step m, n is the number of observed action segments and N is the total number of action segments in the video.

Since the input to the encoder are frame-wise features which might be very long, the output of the encoder $h_{t_o}^e$ might not be able to capture all the relevant information. To alleviate this problem

we combine the decoder with an attention mechanism using a multi-head attention module (*Vaswani et al.*, 2017). Given the current hidden state of the decoder h_m^d and the output of the encoder at each step $h_{1:t_o}^e = (h_1^e, \ldots, h_{t_o}^e)$, the attention output for a single head is computed by first generating the query (q), the keys (K) and values (V) as described in the following

$$q = W_q h_m^d, \tag{8.9}$$

$$K = W_k h_{1:t_o}^e, \tag{8.10}$$

$$V = W_v h_{1:t_o}^e, \tag{8.11}$$

where $q \in \mathbb{R}^{d_A}$ is a column vector, $K,\ V \in \mathbb{R}^{d_A \times t_o}$ and d_A is set to be one-eighth of the hidden state size of the GRU. We normalize these tensors by the $\|.\|_2$ norm and then compute the output using a weighted sum of the values

$$O_h = Softmax(q^T K)V^T, \tag{8.12}$$

where $O_h \in \mathbb{R}^{d_A}$ is the output of a single head. We use 8 heads and concatenate the output of these heads, apply a linear transformation and then concatenate the output with the input of the decoder at each step.

8.2.2 Cycle Consistency

Since predicting the future from the past and the past from the future should be consistent, we propose an additional cycle consistency loss. Given the predicted future activities, we want to predict the past activities and their duration. This requires the predicted future to be good enough to predict the past. The cycle consistency loss has two benefits. First, it verifies if, for the predicted future actions, the actions that have been previously observed are plausible. Second, it encourages the recurrent decoder to keep the most important information of the observed sequence until the end, instead of storing only the information of the previous anticipated activity. In this way, a wrong prediction does not necessary propagate since the observed sequence is kept in memory.

The cycle consistency module is similar to the recurrent decoder and consists of a single layer with GRU, however, it predicts the past instead of the future. The hidden state of this GRU is initialized with the last hidden state of the recurrent decoder and at each step the hidden state is updated as follows

$$h_m^{cyc} = GRU(A_{m-1},\ h_{m-1}^{cyc}). \tag{8.13}$$

Given the hidden state h_m^{cyc} at step m, an activity label of the observations and its relative duration are predicted using a fully connected layer. The loss function is also similar to the recurrent decoder

$$\mathcal{L}_{cyc} = \mathcal{L}_{CE} + \mathcal{L}_{MSE}, \tag{8.14}$$

$$\mathcal{L}_{CE} = \frac{1}{n}\sum_{m=1}^{n} -log(\tilde{a}_{m,A}), \tag{8.15}$$

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{m=1}^{n} (\tilde{\ell}_m - \ell_m)^2, \tag{8.16}$$

where $\mathcal{L}_{cyc}$ is the cycle consistency loss, $\mathcal{L}_{CE}$ and $\mathcal{L}_{MSE}$ are the cross entropy loss and MSE loss applied on the past activity labels and their relative duration.

While the mapping from past to future can be multi-modal, this does not limit the applicability of the cycle consistency module. Since the cycle consistency module is conditioned on the predicted future, no matter what mode is predicted, the cycle consistency makes sure it is plausible. This also applies to the inverse mapping. As there is a path from the observed frames to the cycle consistency module through the sequence-to-sequence model, there is no ambiguity in which past activities have been observed.

8.2.3 Recognition Module

In the sequence-to-sequence model, the input of the recurrent encoder are the frame-wise features. However, directly passing the features to the encoder is sub-optimal as the encoder might struggle to capture all the relevant information for anticipating the future activities. As past activities provide a strong signal for anticipating future activities, we use a recognition module that learns discriminative features of the observed frames. These features will then serve as an input for the sequence-to-sequence model to anticipate the future activities. Given the success of temporal convolutional networks (TCNs) in analyzing activities in videos, we use a similar network for our recognition module. Besides being a strong model for analyzing videos, TCNs are fully differentiable and can be integrated in our framework without preventing end-to-end training. For our module, we use a TCN similar to the one proposed in Chapter 4. The TCN consists of several layers of dilated 1D convolutions where the dilation factor is doubled at each layer. The operations at each layer can be formally described as follows

$$\hat{H}_l = ReLU(W_1 * H_{l-1} + b_1), \tag{8.17}$$

$$H_l = H_{l-1} + W_2 * \hat{H}_l + b_2, \tag{8.18}$$

where H_l is the output of layer l, $*$ is the convolution operator, $W_1 \in \mathbb{R}^{3\times D\times D}$ are the weights of the dilated convolutional filters with kernel size 3 and D is the number of the filters, $W_2 \in \mathbb{R}^{1\times D\times D}$ are the weights of a 1×1 convolution, and $b_1, b_2 \in \mathbb{R}^D$ are bias vectors. The input of the first layer H_0 is obtained by applying a 1×1 convolution over the input features X as follows

$$H_0 = W_0 * X + b_0 \tag{8.19}$$

where $W_0 \in \mathbb{R}^{1\times D\times F}$ and $b_0 \in \mathbb{R}^D$ are the weights and bias for the 1×1 convolutional layer and F is the dimension of the input features.

The output of the last dilated convolutional layer serves as input to the subsequent modules. To make sure that these features are discriminative enough, we add a classification layer that predicts the action label at each observed frame

$$Y_t = Softmax(W h_{L,t} + b), \tag{8.20}$$

where Y_t contains the class probabilities at time t, $h_{L,t} \in \mathbb{R}^D$ is the output of the last dilated convolutional layer at time t, $W \in \mathbb{R}^{C\times D}$ and $b \in \mathbb{R}^C$ are the weights and bias for the 1×1 convolutional layer, where C is the number of action classes. To train this module we use a cross entropy loss

$$\mathcal{L}_R = \frac{1}{t_o} \sum_{t=1}^{t_o} -log(y_{t,a}), \tag{8.21}$$

where $y_{t,a}$ is the predicted probability for the ground truth label a at time t, and t_o is the number of observed frames.

8.2.4 Loss Function

To train our framework, we sum up all the three mentioned losses

$$\mathcal{L} = \mathcal{L}_A + \mathcal{L}_R + \mathcal{L}_{cyc}, \tag{8.22}$$

where $\mathcal{L}_A$ is the anticipation loss, $\mathcal{L}_R$ is the recognition loss, and $\mathcal{L}_{cyc}$ is the cycle consistency loss.

8.3 Experiments

We evaluate the proposed model on two datasets: the Breakfast dataset (*Kuehne et al.*, 2014) and 50Salads (*Stein and McKenna*, 2013). In all experiments, we report the average of three runs.

The **Breakfast** dataset is a collection of $1,712$ videos with overall 66.7 hours and roughly 3.6 million frames. Each video belongs to one out of ten breakfast related activities, such as make tea or pancakes. The video frames are annotated with fine-grained action labels like *pour milk* or *fry egg*. Overall, there are 48 different actions. On average, each video contains 6 action instances and is 2.3 minutes long. For evaluation, we use the standard 4 splits as proposed in (*Kuehne et al.*, 2014) and report the average.

The **50Salads** dataset contains 50 videos showing people preparing different kinds of salad. These videos are relatively long with an average of 6.4 minutes and 20 action instances per video. The video frames are annotated with 17 fine-grained action labels like *cut tomato* or *peel cucumber*. For evaluation, we use five-fold cross-validation and report the average as in (*Stein and McKenna*, 2013).

We follow the state-of-the-art evaluation protocol and report the mean over classes (MoC) accuracy for different observation/prediction percentages.

Implementation Details. For the recognition module, we used a TCN with 10 layers and 64 filters in each layer. The number of units in the GRU cells is set to 512. For each training video, we generate two training examples with 20% and 30% observation percentage. The prediction percentage is always set to 50%. All the models are trained for 80 epochs using Adam optimizer (*Kingma and Ba*, 2015). We set the learning rate to 0.001 and reduce it every 20 epochs with a factor of 0.8. For both datasets, we extract I3D (*Carreira and Zisserman*, 2017) features for the video frames using both RGB and flow streams and sub-sample them at 5 frames per second.

8.3.1 Ablation Analysis

In this section, we analyze the impact of the different modules in our framework on the anticipation performance. This analysis is conducted on the Breakfast dataset and the results are shown in Table 8.1.

Impact of the recognition module: The recognition module consists of two parts: a TCN and a recognition loss $\mathcal{L}_R$. Starting from only the sequence-to-sequence module (S2S), we can achieve a good accuracy in the range $20\% - 27\%$. By combining the sequence-to-sequence module with the recognition module (S2S + TCN + $\mathcal{L}_R$), we gain an improvement of $1\% - 2\%$ for each observation-prediction percentage. This indicates that recognition helps the anticipation task. We also evaluate

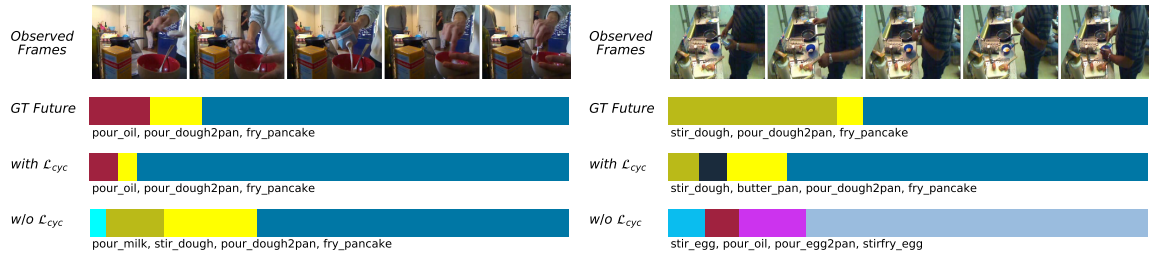
Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
S2S	23.22	20.92	20.10	20.05	26.43	24.38	24.13	23.38
S2S + TCN	14.52	13.55	13.15	12.70	14.83	14.11	13.61	13.01
S2S + TCN + $\mathcal{L}_R$	24.72	22.43	21.70	21.77	28.35	26.29	25.01	24.47
S2S + TCN + $\mathcal{L}_R$ + $\mathcal{L}_{cyc}$	25.16	22.73	22.22	**22.01**	28.07	26.25	25.12	24.81
S2S + TCN + $\mathcal{L}_R$ + $\mathcal{L}_{cyc}$ + attn.	**25.88**	**23.42**	**22.42**	21.54	**29.66**	**27.37**	**25.58**	**25.20**

Table 8.1: Ablation study on the Breakfast dataset. Numbers represent mean over classes (MoC) accuracy.

the performance when the TCN is combined with the sequence-to-sequence module without the recognition loss (S2S + TCN). As shown in Table 8.1, the results are worse than using only the sequence-to-sequence module if we do not apply the recognition loss. This can be explained by the structure of the network. The recurrent encoder maps the features extracted by the TCN from all frames to a single vector. Without additional loss for the recognition module, the gradient vanishes and the parameters of the TCN are not well estimated. Nevertheless, by just applying the recognition loss, we get enough supervisory signal to train the TCN and improve the overall anticipation accuracy. This also highlights that the improvements from the recognition module are due to the additional recognition task and not because of having more parameters.

Impact of the cycle consistency loss: The cycle consistency module predicts the past activities from the predicted future. The intuition is that to be able to predict the past activities, the predicted future activities have to be correct. As shown in Table 8.1, using the cycle consistency loss gives an additional improvement on the anticipation accuracy. The cycle consistency loss verifies if, for the predicted future actions, all required actions have been done before and no essential action is missing. For example in Figure 8.3 (a), the model observes *spoon flour*, *crack egg*, *pour milk*, and *stir dough*. Without the cycle consistency the network did not predict the action *pour oil*, which is required to *fry pancake*. By using cycle consistency this issue is resolved. Cycle consistency also forces the decoder to remember all observed activities. As illustrated in Figure 8.3 (b), the model observes *spoon flour*, *crack egg*, *pour milk*, *butter pan*, and *stir dough*. Without the cycle consistency, the network predicts the action *stirfry egg*, which would have been plausible if *spoon flour* and *pour milk* were not part of the observations. Since the cycle consistency encourages the decoder to use all observed actions for anticipation, the activity *fry pancake* is correctly anticipated.

Impact of the attention module: Finally, using the full model by combining the recurrent decoder with the multi-head attention module further improves the results by roughly 1%. As shown in Table 8.1, the gain from using the attention module is higher when the observation percentage is 30%. This is mainly because of the encoder module. Given the observed frames, the encoder tries to encode them in a single vector. This means that the encoder has to throw away more information from the long sequences compared to shorter sequences. In this case, the attention module can help in capturing some of the lost information in the encoder output by attending on the relevant information in the observations.



(a) (b)

Figure 8.3: Impact of the cycle consistency loss. Cycle consistency verifies if, for the predicted future actions, all required actions have been done before and no essential action is missing (a), and it further encourages the decoder to keep the important information from the observations until the end, which results in better predictions (b).

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Two-Step	23.58	21.90	20.80	20.18	27.80	25.21	23.32	22.96
Ours	**25.88**	**23.42**	**22.42**	**21.54**	**29.66**	**27.37**	**25.58**	**25.20**

Table 8.2: Comparison between the two-step approach and ours on the Breakfast dataset. Numbers represent mean over classes (MoC) accuracy.

8.3.2 End-to-End vs. Two-Step Approach

To illustrate the benefits of end-to-end learning over two-step approaches, we compare our framework with its two-step counterpart. For this comparison, we first train the recognition module from our framework and then fix the weights of the TCN and train the remaining components of our model with the anticipation loss and the cycle consistency loss. Table 8.2 shows the results of our framework compared to the two-step approach on the Breakfast dataset with different observation and prediction percentages. As shown in the table, our framework outperforms the two-step approach with a large margin of up to 2.3%. This highlights the benefits of end-to-end approaches where the model can capture the relevant information in the observed frames to anticipate the future. On the contrary, two-step approaches can only utilize the label information of the observed frames that are not optimized for the anticipation task which is sub-optimal.

8.3.3 Frame-wise Features vs. Frame-wise Labels

In our model, we use the output features of the recognition module as input to the subsequent modules. Since the recognition module also predicts frame-wise class probabilities of the observed frames, another option would be to pass these probabilities instead of the features. In this section, we provide a comparison between these two variants. As shown in Table 8.3, passing frame-wise class probabilities gives a comparable performance to frame-wise features. Nevertheless, using the features is slightly better. This is expected as the features contain much richer information that can be utilized by the subsequent modules. We also tried concatenating the frame-wise features and the

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Ours with class probabilities	25.53	23.39	22.50	**21.70**	28.95	27.36	25.06	24.32
Ours with features	**25.88**	**23.42**	22.42	21.54	**29.66**	**27.37**	**25.58**	**25.20**
Ours with features and prob.	25.15	23.41	**22.53**	21.64	29.21	26.62	24.65	24.69

Table 8.3: Comparison between passing features vs. passing class probabilities of the observed frames from the recognition module to the subsequent modules on the Breakfast dataset. Numbers represent mean over classes (MoC) accuracy.

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Ours with duration input	25.21	20.94	19.61	20.38	28.20	24.71	23.58	23.15
Ours without duration input	**25.88**	**23.42**	**22.42**	**21.54**	**29.66**	**27.37**	**25.58**	**25.20**

Table 8.4: Impact of passing the predicted duration at each step to the recurrent decoder on the Breakfast dataset. Numbers represent mean over classes (MoC) accuracy.

frame-wise class probabilities and pass the concatenated tensor to the sequence-to-sequence module. However, it did not improve the results.

8.3.4 Impact of Passing the Duration to the Decoder

The GRU in the recurrent decoder takes as input at each step the predicted activity label from the previous step as described in (8.2). In this section, we study the impact of passing the predicted duration to the decoder as well, *i.e.*

$$h_m^d = GRU([A_{m-1}, \hat{\ell}_{m-1}],\ h_{m-1}^d). \tag{8.23}$$

As shown in Table 8.4, passing the duration to the decoder results in a degradation in performance. This is due to passing the duration before applying the softmax, which means that there might be huge variations in the duration value between different iterations. While passing the duration after applying the softmax is not feasible, passing only the activity labels provides enough information for the decoder. We also tried to predict the absolute duration. In this case, no EOS symbol is predicted and the decoder keeps generating future activities and their duration until the desired time horizon is predicted. However, the model performed very poorly in this setup. This is expected since predicting a stopping symbol is much easier than predicting the absolute duration of all activities.

8.3.5 Comparison with the State-of-the-Art

In this section, we compare our framework with the state-of-the-art methods on both the Breakfast dataset and 50Salads. We follow the same protocol and report results for different observation and prediction percentages. Table 8.5 shows the results on both datasets. All the previous approaches follow the two-step approach by inferring the action labels of the observed frames first and then use

Observation %	20%				30%			
Prediction %	10%	20%	30%	50%	10%	20%	30%	50%
Breakfast								
RNN model (Chapter 6)	18.11	17.20	15.94	15.81	21.64	20.02	19.73	19.21
CNN model (Chapter 6)	17.90	16.35	15.37	14.54	22.44	20.12	19.69	18.76
RNN (Chapter 6) + MS-TCN	05.93	05.68	05.52	05.11	08.87	08.90	07.62	07.69
RNN (Chapter 6) + MS-TCN*	23.56	20.74	19.52	18.37	26.12	23.55	21.47	20.99
CNN (Chapter 6) + MS-TCN	09.85	09.17	09.06	08.87	17.59	17.13	16.13	14.42
CNN (Chapter 6) + MS-TCN*	25.40	**23.92**	22.09	20.95	28.82	26.29	24.40	22.69
UAAA (mode) (Chapter 7)	16.71	15.40	14.47	14.20	20.73	18.27	18.42	16.86
Time-Cond. (*Ke et al.*, 2019)	18.41	17.21	16.42	15.84	22.75	20.44	19.64	19.75
Ours	**25.88**	23.42	**22.42**	**21.54**	**29.66**	**27.37**	**25.58**	**25.20**
TAR (I3D) (*Sener et al.*, 2020)†	24.20	21.10	20.00	18.10	30.40	26.30	23.80	21.20
TAR (I3D + Labels) (*Sener et al.*, 2020)†	37.10	31.80	30.10	27.10	39.80	34.20	31.90	27.80
50Salads								
RNN model (Chapter 6)	30.06	25.43	18.74	13.49	30.77	17.19	14.79	09.77
CNN model (Chapter 6)	21.24	19.03	15.98	09.87	29.14	20.14	17.46	10.86
RNN (Chapter 6) + MS-TCN	32.31	25.51	19.10	14.15	26.14	17.69	16.33	12.97
RNN (Chapter 6) + MS-TCN*	29.91	24.10	20.48	15.22	28.60	20.09	17.55	12.88
CNN (Chapter 6) + MS-TCN	16.02	14.68	12.09	09.89	19.23	14.68	13.18	11.20
CNN (Chapter 6) + MS-TCN*	15.62	14.48	11.64	9.71	17.16	12.66	11.11	10.55
UAAA (mode) (Chapter 7)	24.86	22.37	19.88	12.82	29.10	20.50	15.28	12.31
Time-Cond. (*Ke et al.*, 2019)	32.51	27.61	21.26	**15.99**	**35.12**	**27.05**	**22.05**	15.59
Ours	**34.76**	**28.41**	**21.82**	15.25	34.39	23.70	18.95	**15.89**
TAR (IDT + Labels) (*Sener et al.*, 2020)†	34.70	25.90	23.70	15.70	34.50	26.10	19.00	15.50
AGG (*Piergiovanni et al.*, 2020)†	40.40	33.70	25.40	20.90	40.70	40.10	26.40	19.20

Table 8.5: Comparison with the state-of-the-art. Numbers represent MoC accuracy. * indicates that MS-TCN is trained with partial sequences that correspond to only the first $15 - 35\%$ of the videos. † indicates concurrent approaches.

these labels to anticipate the future activities. As shown in Table 8.5, our framework outperforms all the previous methods by a large margin of up to 2.5% on the breakfast dataset. For 50Salads, our model outperforms the state-of-the-art in 50% of the cases. This is mainly because 50Salads is a small dataset. Since our model is trained end-to-end, it requires more data to show the benefits over two-step approaches.

On both datasets, the proposed end-to-end framework outperforms both the RNN and the CNN approaches from Chapter 6, even when they are combined with MS-TCN. This highlights that the improvements in our model are not only due to the TCN, but mainly because of the joint optimization of all modules for the anticipation task in an end-to-end fashion and the introduced cycle consistency loss.

Table 8.5 also shows the results of the concurrent approaches (*Sener et al.*, 2020; *Piergiovanni et al.*, 2020). In the work of *Piergiovanni et al.* (2020), an adversarial grammar is learned from the videos of the 50Salads dataset and then used to predict future activities. *Sener et al.* (2020) proposed a framework for aggregating representations over long-video sequences. The learned representations were then used for both recognizing and anticipating activities. For the anticipation experiment, two

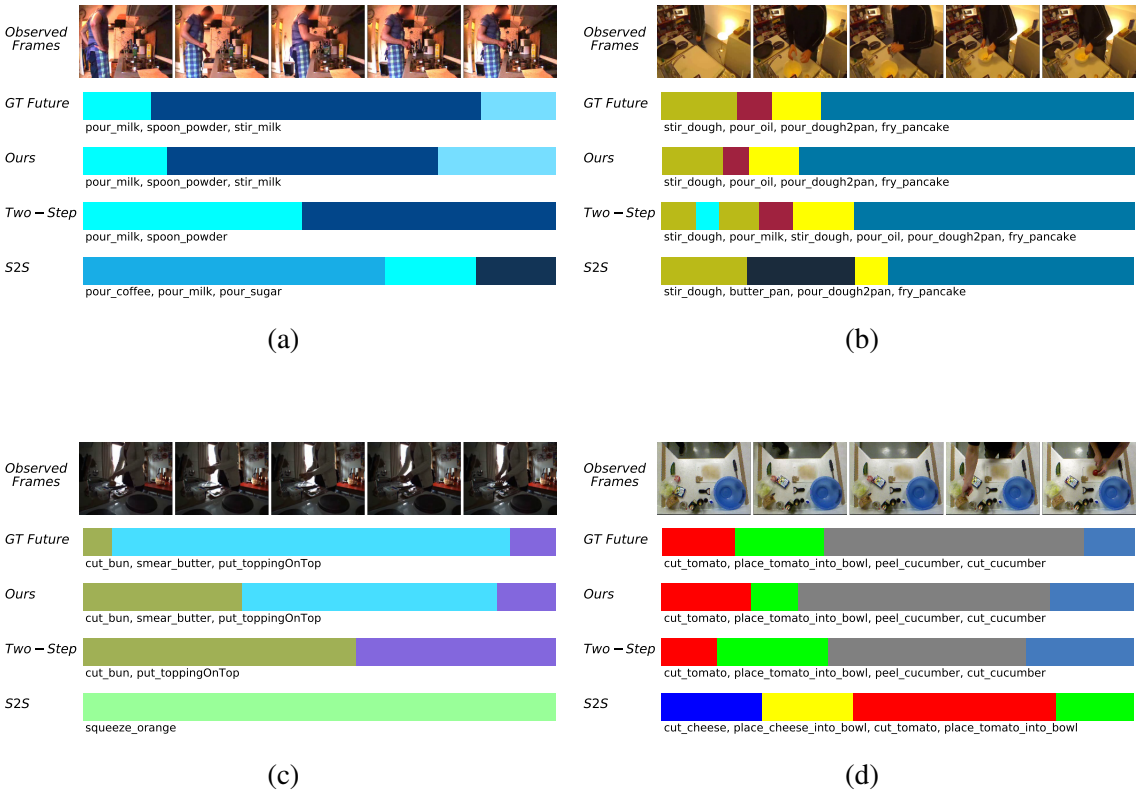


Figure 8.4: Qualitative results for anticipating future activities. (a-c) Three examples from the Breakfast dataset for the case of observing 20% of the videos and predicting the activities in the following 50%. (d) An example from the 50Salads dataset for the case of observing 20% of the video and predicting the activities in the following 20%.

setups were evaluated. The first setup uses only the frame-wise features as input, whereas for the second, the labels of the observed frames are first inferred and then concatenated with the frame-wise features. While the approach of *Sener et al.* (2020) outperforms our framework when applied on the concatenation of the frame-wise action labels and the I3D features on the Breakfast dataset, our framework works better when only the features are used.

Qualitative results for our model on both datasets are illustrated in Figure 8.4. As shown in the figure, our model can generate accurate predictions of the future activities and their duration. We also show the results of the sequence-to-sequence (S2S) and the two-step baselines. Our model anticipates the activities better.

8.4 Summary

In this chapter, we introduced a model for anticipating future activities from a partially observed video. In contrast to the state-of-the-art methods which rely on the action labels of the observations, our model directly predicts the future activities from the observed frames. We train the proposed model in an end-to-end fashion and show a superior performance compared to the previous ap-

proaches. Additionally, we introduced a cycle consistency loss for the anticipation task which further boosts the performance. Our framework achieved state-of-the-art results on two publicly available datasets when the work has been published.

CHAPTER 9

Conclusion

In this chapter, we summarize the contributions of the thesis for both the temporal action segmentation and the anticipation tasks. Furthermore, we discuss future directions for research to advance state-of-the-art approaches for recognizing and anticipating activities in long videos.

Contents

9.1 Summary

In this thesis, we proposed approaches that address the limitations of the current state-of-the-art approaches for both analyzing and anticipating activities in long untrimmed videos with many action segments. First, we proposed two architectures for the temporal action segmentation task that are capable of temporally segmenting activities with high accuracy. Then, we proposed a new level of weak supervision to train these architectures to reduce the required annotation cost. Furthermore, we addressed the action anticipation task and proposed to extend the predicted time horizon up to several minutes into the future. To this end, we introduced several architectures based on RNNs and CNNs that predict sequences of future activities with their durations up to five minutes into the future. Finally, we combined the proposed approaches for both tasks in a unified framework for long-term anticipation of activities. In the following we discuss these contributions in detail.

9.1.1 Temporal Action Segmentation

Previous approaches for temporal action segmentation suffer from several limitations. These approaches either operate on a very limited temporal resolution and have many over-segmentation errors, or rely on heuristics to capture the long-range dependencies such as grammars and length models based on the training data statistics. In Chapter 4, we addressed these limitations and proposed two multi-stage architectures based on temporal convolutional networks. We demonstrated that stacking multiple stages sequentially results in incremental refinement of the predictions. We further showed that having a large receptive field in each stage is crucial for generating accurate action segments. Additionally, we introduced a dual dilated layer that captures both local and global features at each layer, which results in better performance. To reduce over-segmentation errors, we further

introduced a smoothing loss that gives an additional improvement. Our models achieved state-of-the-art results on three challenging datasets and yet they are very efficient and fast both during training and inference.

To reduce the annotation cost required to train action segmentation models, we proposed in Chapter 5 a new level of supervision based on timestamps. In contrast to full supervision, only a single frame from each action segment is annotated for the timestamp supervision. To demonstrate the effectiveness of the proposed level of supervision, we proposed an approach to train an action segmentation model using only timestamps annotations. The proposed approach relies on the model output and the annotated timestamps to generate pseudo-labels by detecting action changes. Furthermore, to reduce the impact of the noisy pseudo-labels, we proposed a confidence loss that enforces monotonicity on the model confidence. This loss ensures that all and not only the most distinctive frames of an action are learned during training by penalizing increases in the model confidence as we move away form the annotations. We showed that models trained with timestamp supervision achieve comparable performance to models that are trained with full supervision.

9.1.2 Long-Term Anticipation of Activities

Previous approaches for action anticipation only predict the next action a few seconds into the future. In this thesis, we proposed to extend the predicted horizon up to several minutes into the future and introduced architectures that can predict multiple activities with their durations. In Chapter 6, we presented two models, a CNN-based model and an RNN-based model, to predict actions in the future for up to five minutes. Both models are trained in a two-step approach by first inferring activities in the observed frames, and then using the proposed models to predict future activities. We demonstrated the ability of the proposed models to generate accurate predictions that scale well along different datasets and videos with varying lengths, varying quality of observed data, and huge variations in the possible future actions. We also showed that activities in the observed frames serve as strong cues to predict future activities.

As the future is multi-modal, generating a single prediction is not enough for many applications. Therefore, we extended our anticipation approach to generate multiple outputs that capture the different possible future activities for an observed sequence of activities. To this end, we proposed in Chapter 7 an action model and a length model based on RNNs to predict a probability distribution over the future action segments. At test time, we sample from the predicted distributions many sequences of future activities.

Finally, we proposed an end-to-end framework for long-term anticipation of activities in Chapter 8. This framework builds on our contributions in Chapter 4 and Chapter 6 and combines the proposed segmentation and anticipation approaches in a unified framework. Furthermore, we introduced a cycle consistency loss for the anticipation task which further boosts the performance. This loss forces the model to predict the past activities form the predicted future activities, which verifies if, for the predicted future actions, all required actions have been done before. Results on two datasets showed that the proposed end-to-end framework outperforms the two-step approaches from Chapter 6 by a large margin.

9.2 Future Work

Research on recognizing and anticipating human activities is still in its early stages. While many of the proposed approaches in this area have been successful, deploying such approaches in real world applications requires addressing their limitations. In this section, we briefly discuss some of the open challenges and possible extensions to the proposed approaches in this thesis.

9.2.1 Advancing Temporal Action Segmentation Approaches

Segment-Wise Loss Functions. The proposed action segmentation models generate frame-wise action probabilities and are trained using frame-wise cross-entropy loss. Although such models perform well on metrics such as the frame-wise accuracy, their performance is poor with respect to segment-based metrics such as the edit and F1 scores. In Chapter 4, we proposed the smoothing loss to address this issue. Nonetheless, the proposed loss is still a frame-based loss and cannot capture the global structure of the predicted segments. While an alternative approach is discussed in *Souri et al.* (2021) by predicting the action segments and their durations instead of the frame-wise representation, the performance of these approaches is significantly worse with respect to frame-based metrics such as the frame-wise accuracy. A promising future direction would investigate segment-based loss functions to train segmentation models that still predict frame-wise probabilities. The challenge, however, is that segment-based metrics are not differentiable with respect to the predicted probabilities. Therefore, it is crucial to formulate differentiable approximations of segment-based metrics and apply them as loss functions.

Principled Receptive Field Design. In Chapter 4, we proposed two architectures for the temporal action segmentation task. While the receptive field in the first architecture is increased exponentially with respect to the number of layers, the second architecture combines both large and small receptive fields at each layer. Despite the success of these approaches, the design choice of the receptive field is purely heuristic based. In *Gao et al.* (2021), a search scheme has been proposed to find better receptive field combinations and it has been shown that the choice of the receptive field can significantly affect the performance. Therefore, it is crucial to investigate and propose principled methods to design the receptive field of the networks.

Handling Noisy Annotations. In Chapter 5, we proposed an approach to train action segmentation models using timestamp annotations. While the proposed approach is robust to noisy annotations, depending on the level of noise the performance can still degrade. An interesting future direction is to explicitly consider annotation errors in the design of the training approach. As a baseline, it is possible to use the model confidence to either ignore segments in the input video, or ignore some of the annotations.

9.2.2 Advancing Long-Term Action Anticipation Approaches

Benchmarking Long-Term Anticipation of Activities. In this thesis, we proposed several approaches to anticipate sequences of future activities. To evaluate these approaches, we did our experiments on datasets with long videos and many action segments (*Kuehne et al.*, 2014; *Stein and McKenna*, 2013). Nonetheless, these datasets are mainly used for the temporal action segmentation

task and not collected with the anticipation task in mind. Therefore, advancing research in long-term action anticipation requires collecting a dedicated dataset for this task. Furthermore, considering different evaluation protocols, different evaluation metrics, and different setups for the amount of observations and predictions, including online prediction, are all important directions for future research.

End-to-End Anticipation with Uncertainty Modelling. In Chapter 8, we proposed an end-to-end model that predicts a sequence of future activities directly from the observed frames. However, this model is deterministic and predicts only a single sequence of activities for a given input of observations. While the uncertainty modelling for long-term anticipation is addressed in Chapter 7, this approach assumes that the anticipation model predicts only one segment at each step and cannot be directly combined with the end-to-end approach that predicts a sequence of activities in one shot. Therefore, it is crucial to address the uncertainty modelling for the end-to-end setup as well.

Adaptive Prediction Horizon. The proposed anticipation approaches in this thesis predict the future for a fixed time horizon. While the RNN approach discussed in Chapter 6 can be applied recursively to anticipate an arbitrarily long time horizon, the prediction made at each step is still for a fixed time horizon. It would be interesting, and beneficial, to design approaches that can adaptively decide on how long into the future it can predict. Such approaches should preferably take uncertainty into consideration for making these decisions. A similar idea has been explored in (*Jayaraman et al.*, 2019) for predicting a single predictable future frame.

Hierarchical Prediction. Human activities are structured hierarchically. The same activity can be described at different levels of abstraction. An interesting future direction is to explore hierarchical models that can predict the future at different levels of granularity. Based on the difficulty of the prediction task, the model has to decide which level to predict. While *Morais et al.* (2020) proposed a hierarchical model that predicts future activities at both coarse and fine levels, there is no mechanism to decide which level is preferable. In (*Surís et al.*, 2021), the selection of the granularity level is addressed in an unsupervised manner by predicting representations in a hyperbolic space. Nonetheless, the approach only works for predicting the next action a few seconds into the future. Combing ideas from these approaches can inspire future research on long-term hierarchical anticipation of activities.

Bibliography

Abu Farha, Yazan and Gall, Juergen. MS-TCN: Multi-stage temporal convolutional network for action segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019a. (Cited on pages 7, 12 and 33.)

Abu Farha, Yazan and Gall, Juergen. Uncertainty-aware anticipation of activities. In *IEEE International Conference on Computer Vision Workshops*, 2019b. (Cited on pages 7, 15 and 89.)

Abu Farha, Yazan; Richard, Alexander, and Gall, Juergen. When will you do what?-Anticipating temporal occurrences of activities. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on pages 3, 7, 15 and 71.)

Abu Farha, Yazan; Ke, Qiuhong; Schiele, Bernt, and Gall, Juergen. Long-term anticipation of activities with cycle consistency. In *DAGM German Conference on Pattern Recognition (GCPR)*, 2020. (Cited on pages 7 and 103.)

Alahi, Alexandre; Goel, Kratarth; Ramanathan, Vignesh; Robicquet, Alexandre; Fei-Fei, Li, and Savarese, Silvio. Social LSTM: Human trajectory prediction in crowded spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on page 16.)

Bearman, Amy; Russakovsky, Olga; Ferrari, Vittorio, and Fei-Fei, Li. What's the point: Semantic segmentation with point supervision. In *European Conference on Computer Vision (ECCV)*, pages 549–565, 2016. (Cited on page 13.)

Bengio, Yoshua; Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. (Cited on page 25.)

Bhattacharya, Subhabrata; Kalayeh, Mahdi M; Sukthankar, Rahul, and Shah, Mubarak. Recognition of complex events: Exploiting temporal dynamics between underlying concepts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2243–2250, 2014. (Cited on page 11.)

Bhattacharyya, Apratim; Fritz, Mario, and Schiele, Bernt. Bayesian prediction of future street scenes using synthetic likelihoods. In *International Conference on Learning Representations (ICLR)*, 2019. (Cited on pages 16 and 94.)

Bilen, Hakan; Fernando, Basura; Gavves, Efstratios; Vedaldi, Andrea, and Gould, Stephen. Dynamic image networks for action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on page 14.)

Bojanowski, Piotr; Lajugie, Rémi; Bach, Francis; Laptev, Ivan; Ponce, Jean; Schmid, Cordelia, and Sivic, Josef. Weakly supervised action labeling in videos under ordering constraints. In *European Conference on Computer Vision (ECCV)*, pages 628–643. Springer, 2014. (Cited on pages 12 and 56.)

Bruckschen, Lilli; Bungert, Kira; Wolter, Moritz; Krumpen, Stefan; Weinmann, Michael; Klein, Reinhard, and Bennewitz, Maren. Where can i help? human-aware placement of service robots. In *IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 2020. (Cited on page 2.)

Buch, Shyamal; Escorcia, Victor; Ghanem, Bernard; Fei-Fei, Li, and Niebles, Juan Carlos. End-to-end, single-stream temporal action detection in untrimmed videos. In *British Machine Vision Conference (BMVC)*, 2017a. (Cited on page 12.)

Buch, Shyamal; Escorcia, Victor; Shen, Chuanqi; Ghanem, Bernard, and Carlos Niebles, Juan. SST: Single-stream temporal action proposals. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017b. (Cited on page 12.)

Carreira, Joao and Zisserman, Andrew. Quo vadis, action recognition? A new model and the kinetics dataset. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017. (Cited on pages 10, 29, 34, 41, 62, 86 and 109.)

Chang, Chien-Yi; Huang, De-An; Sui, Yanan; Fei-Fei, Li, and Niebles, Juan Carlos. D^3TW: Discriminative differentiable dynamic time warping for weakly supervised action alignment and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3546–3555, 2019. (Cited on pages 12 and 68.)

Chao, Yu-Wei; Vijayanarasimhan, Sudheendra; Seybold, Bryan; Ross, David A; Deng, Jia, and Sukthankar, Rahul. Rethinking the faster R-CNN architecture for temporal action localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on page 12.)

Chen, Min-Hung; Li, Baopu; Bao, Yingze; AlRegib, Ghassan, and Kira, Zsolt. Action segmentation with joint self-supervised temporal domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. (Cited on page 52.)

Cheng, Yu; Fan, Quanfu; Pankanti, Sharath, and Choudhary, Alok. Temporal sequence modeling for video event detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2227–2234, 2014. (Cited on page 11.)

Chéron, Guilhem; Alayrac, Jean-Baptiste; Laptev, Ivan, and Schmid, Cordelia. A flexible model for training action localization with varying levels of supervision. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 942–953, 2018. (Cited on page 13.)

Cho, Kyunghyun; van Merriënboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014. (Cited on page 25.)

Damen, Dima; Leelasawassuk, Teesid; Haines, Osian; Calway, Andrew, and Mayol-Cuevas, Walterio. You-do, I-learn: Discovering task relevant objects and their modes of interaction from multi-user egocentric video. In *British Machine Vision Conference (BMVC)*, 2014. (Cited on pages 18, 57, 62 and 68.)

Damen, Dima; Doughty, Hazel; Maria Farinella, Giovanni; Fidler, Sanja; Furnari, Antonino; Kazakos, Evangelos; Moltisanti, Davide; Munro, Jonathan; Perrett, Toby; Price, Will, and others, . Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision (ECCV)*, 2018. (Cited on page 15.)

Dantone, M.; Gall, J.; Leistner, C., and Van Gool, L. Body Parts Dependent Joint Regressors for Human Pose Estimation in Still Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2131–2143, 2014. (Cited on page 37.)

de Bezenac, Emmanuel; Pajot, Arthur, and Gallinari, Patrick. Deep learning for physical processes: Incorporating prior scientific knowledge. In *International Conference on Learning Representations (ICLR)*, 2018. (Cited on page 16.)

Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. (Cited on pages 10 and 29.)

Dessalene, Eadom; Devaraj, Chinmaya; Maynord, Michael; Fermuller, Cornelia, and Aloimonos, Yiannis. Forecasting action through contact representations from first person video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. (Cited on page 15.)

Diba, Ali; Sharma, Vivek, and Van Gool, Luc. Deep temporal linear encoding networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. (Cited on page 10.)

Diba, Ali; Fayyaz, Mohsen; Sharma, Vivek; Arzani, M Mahdi; Yousefzadeh, Rahman; Gall, Juergen, and Van Gool, Luc. Spatio-temporal channel correlation networks for action classification. In *European Conference on Computer Vision (ECCV)*, 2018. (Cited on page 10.)

Ding, Li and Xu, Chenliang. Weakly-supervised action segmentation with iterative soft boundary assignment. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6508–6516, 2018. (Cited on pages xvii, 12, 13, 34, 38, 49, 53 and 68.)

Duchenne, Olivier; Laptev, Ivan; Sivic, Josef; Bach, Francis, and Ponce, Jean. Automatic annotation of human actions in video. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1491–1498, 2009. (Cited on page 12.)

Epstein, Dave; Chen, Boyuan, and Vondrick, Carl. Oops! predicting unintentional action in video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. (Cited on page 16.)

Fathi, Alireza and Rehg, James M. Modeling actions through state changes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2579–2586, 2013. (Cited on page 11.)

Fathi, Alireza; Farhadi, Ali, and Rehg, James M. Understanding egocentric activities. In *IEEE International Conference on Computer Vision (ICCV)*, pages 407–414, 2011a. (Cited on page 11.)

Fathi, Alireza; Ren, Xiaofeng, and Rehg, James M. Learning to recognize objects in egocentric activities. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3281–3288, 2011b. (Cited on pages 17, 36, 41, 57 and 62.)

Fayyaz, Mohsen and Gall, Jurgen. SCT: Set constrained temporal transformer for set supervised action segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. (Cited on pages 13, 56 and 68.)

Fayyaz, Mohsen; Bahrami, Emad; Diba, Ali; Noroozi, Mehdi; Adeli, Ehsan; Van Gool, Luc, and Gall, Jurgen. 3D CNNs with adaptive temporal feature resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on page 10.)

Feichtenhofer, Christoph. X3D: Expanding architectures for efficient video recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. (Cited on page 10.)

Feichtenhofer, Christoph; Pinz, Axel, and Wildes, Richard. Spatiotemporal residual networks for video action recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3468–3476, 2016. (Cited on pages 10 and 34.)

Feichtenhofer, Christoph; Fan, Haoqi; Malik, Jitendra, and He, Kaiming. Slowfast networks for video recognition. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. (Cited on pages 10 and 34.)

Fernando, Basura and Herath, Samitha. Anticipating human actions by correlating past with the future with jaccard similarity measures. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on page 14.)

Furnari, Antonino and Farinella, Giovanni Maria. What would you expect? Anticipating egocentric actions with rolling-unrolling LSTMs and modality attention. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. (Cited on page 15.)

Gammulle, Harshala; Denman, Simon; Sridharan, Sridha, and Fookes, Clinton. Forecasting future action sequences with neural memory networks. In *British Machine Vision Conference (BMVC)*, 2019a. (Cited on pages 15 and 103.)

Gammulle, Harshala; Denman, Simon; Sridharan, Sridha, and Fookes, Clinton. Predicting the future: A jointly learnt model for action anticipation. In *IEEE International Conference on Computer Vision (ICCV)*, 2019b. (Cited on page 14.)

Gao, Jiyang; Yang, Zhenheng; Chen, Kan; Sun, Chen, and Nevatia, Ram. Turn tap: Temporal unit regression network for temporal action proposals. In *IEEE International Conference on Computer Vision (ICCV)*, 2017a. (Cited on page 12.)

Gao, Jiyang; Yang, Zhenheng, and Nevatia, Ram. Cascaded boundary regression for temporal action detection. In *British Machine Vision Conference (BMVC)*, 2017b. (Cited on page 12.)

Gao, Jiyang; Yang, Zhenheng, and Nevatia, Ram. RED: Reinforced encoder-decoder networks for action anticipation. In *British Machine Vision Conference (BMVC)*, 2017c. (Cited on pages 2, 14 and 89.)

Gao, Shang-Hua; Han, Qi; Li, Zhong-Yu; Peng, Pai; Wang, Liang, and Cheng, Ming-Ming. Global2local: Efficient structure search for video action segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on pages 12, 52, 54 and 119.)

Garbade, Martin and Gall, Juergen. Thinking outside the box: Spatial anticipation of semantic categories. In *British Machine Vision Conference (BMVC)*, 2017. (Cited on page 85.)

Girdhar, Rohit and Grauman, Kristen. Anticipative video transformer. *arXiv preprint arXiv:2106.02036*, 2021. (Cited on page 15.)

Graber, Colin; Tsai, Grace; Firman, Michael; Brostow, Gabriel, and Schwing, Alexander G. Panoptic segmentation forecasting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on page 16.)

Guen, Vincent Le and Thome, Nicolas. Disentangling physical dynamics from unknown factors for unsupervised video prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. (Cited on page 16.)

He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. (Cited on pages 10 and 24.)

Hecker, Simon; Dai, Dengxin, and Van Gool, Luc. Failure prediction for autonomous driving. In *IEEE Intelligent Vehicles Symposium (IV)*, 2018. (Cited on page 16.)

Hoai, Minh and De la Torre, Fernando. Max-margin early event detectors. *International Journal of Computer Vision*, 107(2):191–202, 2014. (Cited on pages 14, 72 and 89.)

Hochreiter, Sepp and Schmidhuber, Juergen. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997. (Cited on page 25.)

Huang, De-An; Fei-Fei, Li, and Niebles, Juan Carlos. Connectionist temporal modeling for weakly supervised action labeling. In *European Conference on Computer Vision (ECCV)*, pages 137–153. Springer, 2016. (Cited on pages 12 and 68.)

Huang, Yifei; Sugano, Yusuke, and Sato, Yoichi. Improving action segmentation via graph-based temporal reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14024–14034, 2020. (Cited on pages 12, 52 and 54.)

Ishikawa, Yuchi; Kasai, Seito; Aoki, Yoshimitsu, and Kataoka, Hirokatsu. Alleviating oversegmentation errors by detecting action boundaries. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021. (Cited on pages 12, 52, 54, 68 and 69.)

Jain, Ashesh; Zamir, Amir R.; Savarese, Silvio, and Saxena, Ashutosh. Structural-RNN: Deep learning on spatio-temporal graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on pages 16, 72 and 89.)

Jayaraman, Dinesh; Ebert, Frederik; Efros, Alexei A, and Levine, Sergey. Time-agnostic prediction: Predicting predictable video frames. In *International Conference on Learning Representations (ICLR)*, 2019. (Cited on page 120.)

Karaman, Svebor; Seidenari, Lorenzo, and Del Bimbo, Alberto. Fast saliency based pooling of fisher encoded dense trajectories. In *European Conference on Computer Vision Workshops*, 2014. (Cited on pages 11 and 34.)

Karpathy, Andrej; Toderici, George; Shetty, Sanketh; Leung, Thomas; Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1725–1732, 2014. (Cited on pages 10 and 28.)

Kay, Will; Carreira, Joao; Simonyan, Karen; Zhang, Brian; Hillier, Chloe; Vijayanarasimhan, Sudheendra; Viola, Fabio; Green, Tim; Back, Trevor; Natsev, Paul, and others, . The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. (Cited on page 29.)

Ke, Qiuhong; Fritz, Mario, and Schiele, Bernt. Time-conditioned action anticipation in one shot. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on pages 15, 86, 99, 100, 103 and 113.)

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. (Cited on pages 23, 94 and 109.)

Kitani, Kris M; Ziebart, Brian D; Bagnell, James Andrew, and Hebert, Martial. Activity forecasting. In *European Conference on Computer Vision (ECCV)*, 2012. (Cited on page 16.)

Koppula, Hema S and Saxena, Ashutosh. Anticipating human activities using object affordances for reactive robotic response. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38 (1):14–29, 2016. (Cited on pages 2 and 14.)

Krizhevsky, Alex; Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012. (Cited on page 24.)

Kuehne, H.; Richard, A., and Gall, J. A Hybrid RNN-HMM Approach for Weakly Supervised Temporal Action Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(04):765–779, 2020. (Cited on pages 11, 13 and 53.)

Kuehne, Hilde; Arslan, Ali, and Serre, Thomas. The language of actions: Recovering the syntax and semantics of goal-directed human activities. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 780–787, 2014. (Cited on pages 15, 16, 36, 41, 57, 62, 76, 90, 94, 104, 109 and 119.)

Kuehne, Hilde; Gall, Juergen, and Serre, Thomas. An end-to-end generative framework for video segmentation and recognition. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016. (Cited on pages 11, 28, 34 and 53.)

Kuehne, Hilde; Richard, Alexander, and Gall, Juergen. Weakly supervised learning of actions from transcripts. *Computer Vision and Image Understanding*, 163:78–89, 2017. (Cited on pages 11, 13 and 53.)

Kuehne, Hilde; Richard, Alexander, and Gall, Juergen. A Hybrid RNN-HMM approach for weakly supervised temporal action segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(04):765–779, 2020. (Cited on pages xvii, 13 and 69.)

Lan, Tian; Chen, Tsung-Chuan, and Savarese, Silvio. A hierarchical representation for future action prediction. In *European Conference on Computer Vision (ECCV)*. Springer, 2014. (Cited on pages 2, 14, 72 and 89.)

Laptev, Ivan. On space-time interest points. *International journal of computer vision*, 64(2):107–123, 2005. (Cited on page 9.)

Lea, Colin; Reiter, Austin; Vidal, René, and Hager, Gregory D. Segmental spatiotemporal CNNs for fine-grained action segmentation. In *European Conference on Computer Vision (ECCV)*, pages 36–52. Springer, 2016. (Cited on pages 34 and 53.)

Lea, Colin; Flynn, Michael D.; Vidal, Rene; Reiter, Austin, and Hager, Gregory D. Temporal convolutional networks for action segmentation and detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. (Cited on pages 11, 12, 34, 41, 49, 53 and 71.)

LeCun, Yann; Bottou, Léon; Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (Cited on pages xiii and 24.)

Lei, Peng and Todorovic, Sinisa. Temporal deformable residual networks for action segmentation in videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6742–6751, 2018. (Cited on pages 12, 34, 49, 53 and 71.)

Levenshtein, Vladimir Iosifovich. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. (Cited on page 30.)

Li, Jun and Todorovic, Sinisa. Set-constrained viterbi for set-supervised action segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10820–10829, 2020. (Cited on pages 13, 56 and 68.)

Li, Jun and Todorovic, Sinisa. Anchor-constrained viterbi for set-supervised action segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9806–9815, 2021. (Cited on page 13.)

Li, Jun; Lei, Peng, and Todorovic, Sinisa. Weakly supervised energy-based learning for action segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 6243–6251, 2019. (Cited on pages 12, 56 and 68.)

Li, Shijie; Abu Farha, Yazan; Liu, Yun; Cheng, Ming-Ming, and Gall, Juergen. MS-TCN++: Multi-stage temporal convolutional network for action segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. (Cited on pages 7, 12 and 33.)

Li, Zhe; Abu Farha, Yazan, and Gall, Jurgen. Temporal action segmentation from timestamp supervision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8365–8374, 2021. (Cited on pages 7 and 55.)

Liang, Junwei; Jiang, Lu; Niebles, Juan Carlos; Hauptmann, Alexander G., and Fei-Fei, Li. Peeking into the future: Predicting future person activities and locations in videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on page 14.)

Liang, Xiaodan; Lee, Lisa; Dai, Wei, and Xing, Eric P. Dual motion GAN for future-flow embedded video prediction. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 16.)

Lin, Chuming; Xu, Chengming; Luo, Donghao; Wang, Yabiao; Tai, Ying; Wang, Chengjie; Li, Jilin; Huang, Feiyue, and Fu, Yanwei. Learning salient boundary feature for anchor-free temporal action localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on page 12.)

Lin, Tianwei; Liu, Xiao; Li, Xin; Ding, Errui, and Wen, Shilei. BMN: Boundary-matching network for temporal action proposal generation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 3889–3898, 2019. (Cited on page 12.)

Lin, Tsung-Yi; Dollár, Piotr; Girshick, Ross; He, Kaiming; Hariharan, Bharath, and Belongie, Serge. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. (Cited on page 38.)

Liu, Daochang; Li, Qiyue; Jiang, Tingting; Wang, Yizhou; Miao, Rulin; Shan, Fei, and Li, Ziyu. Towards unified surgical skill assessment. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on page 2.)

Liu, Miao; Tang, Siyu; Li, Yin, and Rehg, James M. Forecasting human-object interaction: joint prediction of motor attention and actions in first person video. In *European Conference on Computer Vision (ECCV)*, 2020. (Cited on page 15.)

Long, Fuchen; Yao, Ting; Qiu, Zhaofan; Tian, Xinmei; Luo, Jiebo, and Mei, Tao. Gaussian temporal awareness networks for action localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on page 12.)

Luc, Pauline; Neverova, Natalia; Couprie, Camille; Verbeek, Jakob, and LeCun, Yann. Predicting deeper into the future of semantic segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 16.)

Luc, Pauline; Couprie, Camille; Lecun, Yann, and Verbeek, Jakob. Predicting future instance segmentation by forecasting convolutional features. In *European Conference on Computer Vision (ECCV)*, 2018. (Cited on page 16.)

Ma, Fan; Zhu, Linchao; Yang, Yi; Zha, Shengxin; Kundu, Gourab; Feiszli, Matt, and Shou, Zheng. SF-Net: Single-frame supervision for temporal action localization. In *European Conference on Computer Vision (ECCV)*, 2020. (Cited on pages xvii, 13, 56, 58, 67, 68, 69 and 70.)

Ma, Shugao; Sigal, Leonid, and Sclaroff, Stan. Learning activity progression in LSTMs for activity detection and early detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on pages 14, 72 and 89.)

Mac, Khoi-Nguyen C.; Joshi, Dhiraj; Yeh, Raymond A.; Xiong, Jinjun; Feris, Rogerio S., and Do, Minh N. Learning motion in feature space: Locally-consistent deformable convolution networks for fine-grained action detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. (Cited on pages 12 and 53.)

Mahmud, Tahmida; Hasan, Mahmudul, and Roy-Chowdhury, Amit K. Joint prediction of activity labels and starting times in untrimmed videos. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on pages 14 and 72.)

Mahmud, Tahmida; Billah, Mohammad; Hasan, Mahmudul, and Roy-Chowdhury, Amit K. Prediction and description of near-future activities in video. *Computer Vision and Image Understanding*, page 103230, 2021. (Cited on pages 2 and 16.)

Martinez, Julieta; Black, Michael J, and Romero, Javier. On human motion prediction using recurrent neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. (Cited on page 16.)

Mathieu, Michael; Couprie, Camille, and LeCun, Yann. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations (ICLR)*, 2016. (Cited on pages 1 and 16.)

Mehrasa, Nazanin; Jyothi, Akash Abdu; Durand, Thibaut; He, Jiawei; Sigal, Leonid, and Mori, Greg. A variational auto-encoder model for stochastic point processes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on pages xviii, 14, 94, 99 and 100.)

Meng, Yue; Lin, Chung-Ching; Panda, Rameswar; Sattigeri, Prasanna; Karlinsky, Leonid; Oliva, Aude; Saenko, Kate, and Feris, Rogerio. AR-Net: Adaptive frame resolution for efficient action recognition. In *European Conference on Computer Vision (ECCV)*, 2020. (Cited on page 10.)

Mettes, Pascal; Van Gemert, Jan C, and Snoek, Cees GM. Spot on: Action localization from pointly-supervised proposals. In *European Conference on Computer Vision (ECCV)*, pages 437–453. Springer, 2016. (Cited on page 13.)

Miech, Antoine; Laptev, Ivan; Sivic, Josef; Wang, Heng; Torresani, Lorenzo, and Tran, Du. Leveraging the present to anticipate the future in videos. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019. (Cited on page 15.)

Moltisanti, Davide; Fidler, Sanja, and Damen, Dima. Action recognition from single timestamp supervision in untrimmed videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9915–9924, 2019. (Cited on pages 13, 56, 67, 68 and 69.)

Morais, Romero; Le, Vuong; Tran, Truyen, and Venkatesh, Svetha. Learning to abstract and predict human actions. In *British Machine Vision Conference (BMVC)*, 2020. (Cited on pages 15 and 120.)

Nabavi, Seyed Shahabeddin; Rochan, Mrigank, and Wang, Yang. Future semantic segmentation with convolutional LSTM. In *British Machine Vision Conference (BMVC)*, 2018. (Cited on page 16.)

Newell, Alejandro; Yang, Kaiyu, and Deng, Jia. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision (ECCV)*, 2016. (Cited on page 37.)

Oneata, Dan; Verbeek, Jakob, and Schmid, Cordelia. The lear submission at THUMOS 2014. 2014. (Cited on page 34.)

Öztürk, Halil İbrahim and Can, Ahmet Burak. Adnet: Temporal anomaly detection in surveillance videos. In *Pattern Recognition. ICPR International Workshops and Challenges*, 2021. (Cited on page 2.)

Paszke, Adam; Gross, Sam; Chintala, Soumith; Chanan, Gregory; Yang, Edward; DeVito, Zachary; Lin, Zeming; Desmaison, Alban; Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems Workshops*, 2017. (Cited on page 94.)

Pei, Mingtao; Si, Zhangzhang; Yao, Benjamin Z, and Zhu, Song-Chun. Learning and parsing video events with goal and intent prediction. *Computer Vision and Image Understanding*, 117(10):1369–1383, 2013. (Cited on page 72.)

Piergiovanni, AJ; Angelova, Anelia; Toshev, Alexander, and Ryoo, Michael S. Adversarial generative grammars for human activity prediction. In *European Conference on Computer Vision (ECCV)*, 2020. (Cited on pages 15, 86, 87 and 113.)

Qiu, Zhaofan; Yao, Ting, and Mei, Tao. Learning spatio-temporal representation with pseudo-3d residual networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 10.)

Ranzato, MarcAurelio; Szlam, Arthur; Bruna, Joan; Mathieu, Michael; Collobert, Ronan, and Chopra, Sumit. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014. (Cited on pages 1 and 16.)

Rasouli, Amir; Kotseruba, Iuliia, and Tsotsos, John K. Pedestrian action anticipation using contextual feature fusion in stacked RNNs. In *British Machine Vision Conference (BMVC)*, 2019. (Cited on page 16.)

Renz, Katrin; Stache, Nicolaj C; Albanie, Samuel, and Varol, Gül. Sign language segmentation with temporal convolutional networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021. (Cited on page 2.)

Richard, Alexander and Gall, Juergen. Temporal action detection using a statistical language model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3131–3140, 2016. (Cited on pages 11 and 53.)

Richard, Alexander; Kuehne, Hilde, and Gall, Juergen. Weakly supervised action learning with RNN based fine-to-coarse modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. (Cited on pages 11, 13, 34, 53, 68, 71, 73, 77, 78, 80, 83, 91 and 97.)

Richard, Alexander; Kuehne, Hilde, and Gall, Juergen. Action sets: Weakly supervised action segmentation without ordering constraints. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5996, 2018a. (Cited on pages 13, 56 and 68.)

Richard, Alexander; Kuehne, Hilde; Iqbal, Ahsan, and Gall, Juergen. NeuralNetwork-Viterbi: A framework for weakly supervised video learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7386–7395, 2018b. (Cited on pages 12, 56 and 68.)

Rivoir, Dominik; Bodenstedt, Sebastian; Funke, Isabel; von Bechtolsheim, Felix; Distler, Marius; Weitz, Jürgen, and Speidel, Stefanie. Rethinking anticipation tasks: Uncertainty-aware anticipation of sparse surgical instrument usage for context-aware assistance. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2020. (Cited on pages 2 and 16.)

Rodriguez, Cristian; Fernando, Basura, and Li, Hongdong. Action anticipation by predicting future dynamic images. In *European Conference on Computer Vision Workshops*. Springer, 2018. (Cited on page 14.)

Rohrbach, Marcus; Amin, Sikandar; Andriluka, Mykhaylo, and Schiele, Bernt. A database for fine grained activity detection of cooking activities. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1194–1201, 2012. (Cited on pages 11 and 34.)

Ruiz, Alejandro Hernandez; Gall, Juergen, and Moreno-Noguer, Francesc. Human motion prediction via spatio-temporal inpainting. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. (Cited on page 16.)

Ryoo, Michael S. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *IEEE International Conference on Computer Vision (ICCV)*, 2011. (Cited on pages 14, 72 and 89.)

Sadegh Aliakbarian, Mohammad; Sadat Saleh, Fatemeh; Salzmann, Mathieu; Fernando, Basura; Petersson, Lars, and Andersson, Lars. Encouraging LSTMs to anticipate actions very early. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on pages 14, 72 and 89.)

Sánchez, Jorge; Perronnin, Florent; Mensink, Thomas, and Verbeek, Jakob. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013. (Cited on pages 9 and 28.)

Sener, Fadime and Yao, Angela. Zero-shot anticipation for instructional activities. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. (Cited on pages 2 and 16.)

Sener, Fadime; Singhania, Dipika, and Yao, Angela. Temporal aggregate representations for long-range video understanding. In *European Conference on Computer Vision (ECCV)*, 2020. (Cited on pages 15, 86, 87, 113 and 114.)

Shi, Yuge; Fernando, Basura, and Hartley, Richard. Action anticipation with RBF kernelized feature mapping RNN. In *European Conference on Computer Vision (ECCV)*, 2018. (Cited on pages 2 and 14.)

Shou, Zheng; Wang, Dongang, and Chang, Shih-Fu. Temporal action localization in untrimmed videos via multi-stage CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on page 12.)

Simonyan, Karen and Zisserman, Andrew. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 568–576, 2014. (Cited on pages 10, 28 and 34.)

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. (Cited on pages 24 and 84.)

Singh, Bharat; Marks, Tim K; Jones, Michael; Tuzel, Oncel, and Shao, Ming. A multi-stream bi-directional recurrent neural network for fine-grained action detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1961–1970, 2016a. (Cited on pages 11 and 53.)

Singh, Suriya; Arora, Chetan, and Jawahar, C. V. First person action recognition using deep learned descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2620–2628, 2016b. (Cited on page 11.)

Soran, Bilge; Farhadi, Ali, and Shapiro, Linda. Generating notifications for missing actions: Don't forget to turn the lights off! In *IEEE International Conference on Computer Vision (ICCV)*, 2015. (Cited on page 2.)

Souri, Yaser; Fayyaz, Mohsen; Minciullo, Luca; Francesca, Gianpiero, and Gall, Juergen. Fast weakly supervised action segmentation using mutual consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. (Cited on pages 13, 52, 54, 68 and 119.)

Srivastava, Nitish; Mansimov, Elman, and Salakhudinov, Ruslan. Unsupervised learning of video representations using LSTMs. In *International Conference on Machine Learning (ICML)*, pages 843–852, 2015. (Cited on page 16.)

Stein, Sebastian and McKenna, Stephen J. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 729–738, 2013. (Cited on pages 16, 36, 41, 57, 62, 76, 90, 94, 104, 109 and 119.)

Sun, Chen; Shrivastava, Abhinav; Vondrick, Carl; Sukthankar, Rahul; Murphy, Kevin, and Schmid, Cordelia. Relational action forecasting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019a. (Cited on page 14.)

Sun, Jiangxin; Xie, Jiafeng; Hu, Jian-Fang; Lin, Zihang; Lai, Jianhuang; Zeng, Wenjun, and Zheng, Wei-Shi. Predicting future instance segmentation with contextual pyramid convLSTMs. In *ACM International Conference on Multimedia*, 2019b. (Cited on page 16.)

Surís, Dídac; Liu, Ruoshi, and Vondrick, Carl. Learning the predictability of the future. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on pages 14 and 120.)

Suzuki, Tomoyuki; Kataoka, Hirokatsu; Aoki, Yoshimitsu, and Satoh, Yutaka. Anticipating traffic accidents with adaptive loss and large-scale incident DB. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on page 16.)

Tai, Tsung-Ming; Fiameni, Giuseppe; Lee, Cheng-Kuang, and Lanz, Oswald. Higher order recurrent space-time transformer. *arXiv preprint arXiv:2104.08665*, 2021. (Cited on page 15.)

Tang, Kevin; Fei-Fei, Li, and Koller, Daphne. Learning latent temporal structure for complex event detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1250–1257, 2012. (Cited on page 11.)

Tran, Du; Bourdev, Lubomir; Fergus, Rob; Torresani, Lorenzo, and Paluri, Manohar. Learning spatiotemporal features with 3d convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. (Cited on page 10.)

Tran, Du; Wang, Heng; Torresani, Lorenzo; Ray, Jamie; LeCun, Yann, and Paluri, Manohar. A closer look at spatiotemporal convolutions for action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on page 10.)

Tran, Vinh; Wang, Yang, and Hoai, Minh. Back to the future: Knowledge distillation for human action anticipation. *arXiv preprint arXiv:1904.04868*, 2019. (Cited on page 14.)

Van Den Oord, Aäron; Dieleman, Sander; Zen, Heiga; Simonyan, Karen; Vinyals, Oriol; Graves, Alex; Kalchbrenner, Nal; Senior, Andrew W, and Kavukcuoglu, Koray. Wavenet: A generative model for raw audio. In *ISCA Speech Synthesis Workshop (SSW)*, 2016. (Cited on pages 11 and 36.)

Varol, Gül; Laptev, Ivan, and Schmid, Cordelia. Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1510–1517, 2018. (Cited on page 10.)

Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz, and Polosukhin, Illia. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. (Cited on page 107.)

Vats, Kanav; Fani, Mehrnaz; Walters, Pascale; Clausi, David A, and Zelek, John. Event detection in coarsely annotated sports videos via parallel multi-receptive field 1D convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 882–883, 2020. (Cited on pages 62 and 66.)

Villegas, Ruben; Yang, Jimei; Zou, Yuliang; Sohn, Sungryull; Lin, Xunyu, and Lee, Honglak. Learning to generate long-term future via hierarchical prediction. In *International Conference on Machine Learning (ICML)*, 2017. (Cited on page 16.)

Vo, Nam N and Bobick, Aaron F. From stochastic grammar to bayes network: Probabilistic parsing of complex activity. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2641–2648, 2014. (Cited on page 11.)

Vondrick, Carl; Pirsiavash, Hamed, and Torralba, Antonio. Anticipating visual representations from unlabeled video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on pages xviii, 2, 14, 72, 85, 86 and 89.)

Walker, Jacob; Marino, Kenneth; Gupta, Abhinav, and Hebert, Martial. The pose knows: Video forecasting by generating pose futures. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 16.)

Wang, Heng and Schmid, Cordelia. Action recognition with improved trajectories. In *IEEE International Conference on Computer Vision (ICCV)*, 2013. (Cited on pages 9 and 27.)

Wang, Heng; Kläser, Alexander; Schmid, Cordelia, and Liu, Cheng-Lin. Action recognition by dense trajectories. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. (Cited on page 9.)

Wang, Heng; Kläser, Alexander; Schmid, Cordelia, and Liu, Cheng-Lin. Dense trajectories and motion boundary descriptors for action recognition. *International Journal of Computer Vision*, 103(1):60–79, 2013. (Cited on pages xiii, 9, 27 and 28.)

Wang, Limin; Xiong, Yuanjun; Wang, Zhe; Qiao, Yu; Lin, Dahua; Tang, Xiaoou, and Van Gool, Luc. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision (ECCV)*, 2016. (Cited on pages 10 and 15.)

Wang, Limin; Tong, Zhan; Ji, Bin, and Wu, Gangshan. TDN: Temporal difference networks for efficient action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (Cited on page 10.)

Wang, Xiaolong; Girshick, Ross; Gupta, Abhinav, and He, Kaiming. Non-local neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on page 10.)

Wang, Zhenzhi; Gao, Ziteng; Wang, Limin; Li, Zhifeng, and Wu, Gangshan. Boundary-aware cascade networks for temporal action segmentation. In *European Conference on Computer Vision (ECCV)*, 2020. (Cited on pages 12, 52, 54, 68 and 69.)

Wei, Shih-En; Ramakrishna, Varun; Kanade, Takeo, and Sheikh, Yaser. Convolutional pose machines. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4732, 2016. (Cited on page 37.)

Wu, Chao-Yuan; Feichtenhofer, Christoph; Fan, Haoqi; He, Kaiming; Krahenbuhl, Philipp, and Girshick, Ross. Long-term feature banks for detailed video understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on page 10.)

Wu, Tz-Ying; Chien, Ting-An; Chan, Cheng-Sheng; Hu, Chan-Wei, and Sun, Min. Anticipating daily intention using on-wrist motion triggered sensing. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 16.)

Xu, Huijuan; Das, Abir, and Saenko, Kate. R-C3D: Region convolutional 3d network for temporal activity detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 12.)

Yeung, Serena; Russakovsky, Olga; Mori, Greg, and Fei-Fei, Li. End-to-end learning of action detection from frame glimpses in videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on page 12.)

Zatsarynna, Olga; Abu Farha, Yazan, and Gall, Juergen. Multi-modal temporal convolutional network for anticipating actions in egocentric videos. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2021. (Cited on page 15.)

Zeng, Kuo-Hao; Shen, William B; Huang, De-An; Sun, Min, and Carlos Niebles, Juan. Visual forecasting by imitating dynamics in natural sequences. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 14.)

Zeng, Runhao; Huang, Wenbing; Tan, Mingkui; Rong, Yu; Zhao, Peilin; Huang, Junzhou, and Gan, Chuang. Graph convolutional networks for temporal action localization. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. (Cited on page 12.)

Zhao, He and Wildes, Richard P. On diverse asynchronous activity anticipation. In *European Conference on Computer Vision (ECCV)*, 2020. (Cited on pages xviii, 15, 99 and 100.)

Zhao, Yue; Xiong, Yuanjun; Wang, Limin; Wu, Zhirong; Tang, Xiaoou, and Lin, Dahua. Temporal action detection with structured segment networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (Cited on page 12.)

Zhou, Bolei; Andonian, Alex; Oliva, Aude, and Torralba, Antonio. Temporal relational reasoning in videos. In *European Conference on Computer Vision (ECCV)*, 2018. (Cited on page 10.)