AN ABSTRACT OF THE DISSERTATION OF

Peng Lei for the degree of Doctor of Philosophy in Computer Science presented on December 7, 2018.

Title: Pixel- and Frame-level Video Labeling using Spatial and Temporal Convolutional Networks

Abstract approved: _____

Sinisa Todorovic

This dissertation addresses the problem of video labeling at both the frame and pixel levels using deep learning. For pixel-level video labeling, we have studied two problems: i) Spatiotemporal video segmentation and ii) Boundary detection and boundary flow estimation. For the problem of spatiotemporal video segmentation, we have developed recurrent temporal deep field (RTDF). RTDF is a conditional random field (CRF) that combines a deconvolution neural network and a recurrent temporal restricted Boltzmann machine (RTRBM), which can be jointly trained end-to-end. We have derived a mean-field inference algorithm to jointly predict all latent variables in both RTRBM and CRF. For the problem of boundary detection and boundary flow estimation, we have proposed a fully convolutional Siamese network (FCSN). The FCSN first estimates object boundaries in two consecutive frames, and then predicts boundary correspondences in the two frames. For frame-level video labeling, we have specified a temporal deformable residual network (TDRN) for temporal action segmentation. TDRN computes two parallel temporal processes: i) Residual stream that analyzes video information at its full temporal resolution, and ii) Pooling/unpooling stream that captures long-range visual cues. The former facilitates local, fine-scale action segmentation, and the latter uses multiscale context for improving the accuracy of frame classification. All of our networks have been empirically evaluated on challenging benchmark datasets and compared with the state of the art. Each of the above approaches has outperformed the state of the art at the time of our evaluation.

Pixel- and Frame-level Video Labeling using Spatial and Temporal Convolutional Networks

by

Peng Lei

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented December 7, 2018
Commencement June 2019

Doctor of Philosophy dissertation of Peng Lei presented on December 7, 2018.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Peng Lei, Author

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Professor Sinisa Todorovic, for all his help guiding me through my Ph.D. journey at Oregon State University. He has been encouraging, supportive and patient throughout my graduate study. I learned a lot from him, including, but not limited to, specific algorithms, academic writing and critical thinking skills, and the way to approach a research problem.

I would also like to express my appreciation to other thesis committee members, Professor Fuxin Li, Professor Raviv Raich and Professor Xiaoli Fern. I thank Professor Fuxin Li for his insightful discussions and professional advices during my research. I thank Professor Raviv Raich and Professor Xiaoli Fern for taking their valuable time to teach useful classes and review my dissertation.

I would also like to thank my friends and colleagues at Oregon State University: Mohamed Amer, Sheng Chen, Liping Liu, Jun Li, Anirban Roy, Behrooz Mahasseni, Michael Lam, Dimitrios Trigkakis, Xinze Guan, Xu Hu, Qingkai Lu, Zhongyuan Feng, Xu Xu, Zeyu You, Yuanli Pei, Khoi Nguyen and Liqiang He, as well as other researchers with whom I worked together at Lotus Hill Institute: Tianfu Wu, Zhenyu Yao, Jiangen Zhang, Xiaohan Nie, Bo Li, Xi Song and Yi Xie.

Last but not least, I appreciate my parents for their continuous and unconditional support, and thank my wife, Yao Deng, for her love and understanding.

TABLE OF CONTENTS

Page

1 Introduction 1

2 Recurrent Temporal Deep Field for Semantic Video Labeling 5

2.1 Introduction . . . 5

2.2 Related Work . . . 7

2.3 Recurrent Temporal Deep Field . . . 8

2.3.1 A Brief Review of Restricted Boltzmann Machine . . . 11

2.3.2 A Brief Review of RTRBM . . . 11

2.3.3 DeconvNet . . . 12

2.4 Inference of RTDF . . . 13

2.5 Learning . . . 15

2.6 Results . . . 16

2.6.1 Quantitative Results . . . 18

2.6.2 Qualitative Evaluation . . . 19

2.7 Summary . . . 22

3 Boundary Flow: A Siamese Network that Predicts Boundary Motion without Training on Motion 23

3.1 Introduction . . . 23

3.2 Related Work . . . 26

3.3 Boundary Flow . . . 27

3.3.1 Definition of Boundary Flow . . . 27

3.3.2 Fully Convolutional Siamese Network . . . 28

3.3.3 Boundary Flow Estimation . . . 29

3.4 Training . . . 35

3.5 Results . . . 36

3.5.1 Boundary Detection . . . 36

3.5.2 Boundary Flow Estimation . . . 37

3.5.3 Dense Optical Flow Estimation . . . 38

3.6 Summary . . . 39

4 Temporal Deformable Residual Networks for Action Segmentation in Videos 41

4.1 Introduction . . . 41

TABLE OF CONTENTS (Continued)

Page

4.2 Related Work 43
4.3 Temporal Deformable Residual Network 45
4.3.1 Deformable Temporal Residual Module 47
4.3.2 Deformable Temporal Convolution Module 47
4.4 Network Configurations and Training 50
4.5 Experimental Results 51
4.5.1 Datasets, Metrics and Baselines 52
4.5.2 Comparison with Convolution Models 54
4.5.3 Comparison with the State of the Art 54
4.5.4 Effect of Kernel Size and Network Depth 57
4.6 Summary 58

5 Conclusion 59

Bibliography 60

Appendices 72
A Derivation of Mean-field Updating Equations 73
B Derivation of the Joint Training of RTDF Parameters 75

LIST OF FIGURES

Figure Page

1.1 Gien a video, assign an object class label to every pixel in each video frame. 2

1.2 Given two consecutive video frames, detect object boundaries and estimate motions of object boundaries. 3

1.3 Given a video, label video frames with appropriate action classes. 4

2.1 (a) Our semantic labeling for a Youtube Face video [125] using RTDF. Given a frame at time t, RTDF uses a CRF to fuse both local and long-range spatiotemporal cues for labeling pixels in frame t. The local cues (red box) are extracted by DeconvNet [4] using only pixels of frame t. The long-range spatiotemporal cues (blue box) are estimated by RTRBM [114] (precisely, the hidden layer of RTDF) using a sequence of previous RTDF predictions for pixels in frames $t-1, t-2, \ldots, t-\gamma$. (b) An illustration of RTDF with pixel labels $\mathbf{y}^t$ in frame t, unary potentials $\mathbf{x}^t$, and top two layers $\mathbf{r}^{t-1}$ and $\mathbf{h}^t$ belonging to RTRBM. The high-order potential is distributed to all pixels in frame t via the full connectivity of layers $\mathbf{h}^t$ and $\mathbf{y}^t$, and layers $\mathbf{r}^{t-1}$ and $\mathbf{h}^t$. 6

2.2 Our RTDF is an energy-based model that predicts pixel labels $\mathbf{y}^t$ for frame t, given the unary potential $\mathbf{x}^t$ of DeconvNet, the pairwise potential between neighboring pixel labels in frame t, and the high-order potential defined in terms of $\mathbf{z}^t$, $\mathbf{h}^t$ and $\mathbf{r}^{t-1}$ of RTRBM. The figure shows the time-unfolded visualization of computational processes in RTRBM. RTRBM takes as input previous RTDF predictions $\{\mathbf{y}^{t-1}, \ldots, \mathbf{y}^{t-\gamma}\}$ and encodes the long-range and high-order dependencies through latent variables $\mathbf{r}^{t-1}$. The high-order potential is further distributed to all pixels in frame t via a deterministic mapping (vertical dashed lines) between $\mathbf{y}^t$ and $\mathbf{z}^t$. 9

2.3 Key steps of the mean-field inference overlaid over RTDF which is depicted as in Figure 2.1b. (a) Initialization of $\boldsymbol{\mu}^{(0)}$. (b) Initialization of $\boldsymbol{\nu}^{(0)}$. (c) Updating of $\boldsymbol{\mu}^{(k+1)}$. (d) Updating of $\boldsymbol{\nu}^{(k+1)}$. The red arrows show the information flow. 14

2.4 Frame samples from CamVid. The rows correspond to original images, ground truth, SegNet [4], and RTDF. 20

2.5 Sequence of frames from a sample CamVid video. The rows correspond to input frames and RTDF outputs. 21

LIST OF FIGURES (Continued)

Figure Page

2.6 Frame sequences from two CamVid video clips. The rows correspond to original video frames, ground truth, STRF[55], and RTDF. 22

3.1 Boundary flow estimation. Given two images (a), our approach *jointly*: predicts object boundaries in both images (b), and estimates motion of the boundaries in the two images (c). For clarity, only a part of boundary matches are shown in (c). 23

3.2 FCSN consists of a Siamese encoder and a Siamese decoder and takes two images as input. The two Siamese soft-max outputs of the decoder produce boundary predictions in each of the two input images. Also, the decoder associates the two Siamese branches via the decoder layers and the JFR layer (the green cube) for calculating the excitation attention score, which in turn is used for BF estimation, as indicated by the cyan and purple arrows. The convolution, pooling, softmax and concatenation layers are marked with black, blue, red and brown respectively. Best viewed in color. 26

3.3 Figure 3.3(a) shows the case when a boundary B_1 in frame t is occluded at time $t+1$. Figure 3.3(b) shows the case when a boundary B_1 in frame t is no longer a boundary at time $t+1$ but its pixels are visible. In both cases BF is well-defined and always resides on the boundary. 28

3.4 (a) Estimation of the excitation attention score in frame $t+1$ (bottom) for a particular boundary point in frame t (top; the point is indicated by the arrow). The attention map is well-aligned with the corresponding boundary in frame $t+1$, despite significant motion. (b) Visualization of attention maps at different layers of the decoders of FCSN along the excitation path (cyan) from a particular boundary point in frame t to frame $t+1$ via the JFR. For simplicity, we only show the attention maps in some of the layers from the decoder branch at time t and $t+1$. As can be seen, starting from a pixel on the predicted boundary in frame t, the attention map gradually becomes coarser along the path to the JFR. Then from the JFR to boundary prediction in frame $t+1$, the excitation attention scores gradually become refined and more focused on the most relevant pixels in frame $t+1$. (Best viewed in color) 30

LIST OF FIGURES (Continued)

Figure Page

3.5 Overview of edgelet matching. The matching process consists of three phases: superpixel generation, edgelet matching, and flow placement. The two frames are first over-segmented into large superpixels using the FCSN boundaries. (a) most of the boundary points (in red color) are well aligned with the superpixel boundaries (in cyan color); (b) Example edgelet matches. In the second case, it can be seen clearly that the appearance only matches on one side of the edgelet. (c) The process of matching and flow placement. Sometimes, because of the volatility of edge detection, $\mathbf{x}_t$ and $\mathbf{y}_{t+1}$ falls on different sides of the boundary, we will need to then move $\mathbf{x}_t$ so that they fall on the same side. Note that s_1 and s_2, $s_1^{'}$ and $s_2^{'}$ denote the superpixel pairs falling on the two sides of the edgelets. 32

3.6 Example results on VSB100. In each row from left to right we present (a) input image, (b) ground truth annotation, (c) edge detection [24], (d) object contour detection [130] and (e) our boundary detection. 33

3.7 PR curve for object boundary detection on VSB100. 33

3.8 PR curve for object boundary detection on VSB100 with fine-tuning on both BSDS500 and VSB100 training sets. 34

3.9 Overview of augmenting boundary flow into the framework of CPM-Flow. Given two images, we compute the standard input to CPM-Flow: matches using CPM matching [49] and the edges of the first image using SE [24]. Then we augment the matches with our predicted boundary flow (i.e., matches on the boundaries), as indicated by black arrows. 37

3.10 Example results on MPI-Sintel test dataset. The columns correspond to original images, ground truth, CPM-AUG (i.e., our approach), CPM-Flow [49] and EpicFlow[95]. The rectangles highlight the improvements and the numbers indicate the EPEs. 38

LIST OF FIGURES (Continued)

Figure Page

4.1 For action segmentation, TDRN takes frame-level CNN features as input and outputs frame-wise action labels. TDRN computes two processing streams: Residual stream (marked red) that analyzes video information at its full temporal resolution for precise action segmentation, and Pooling/unpooling stream (marked blue) that captures temporal context at different scales for accurate action recognition. The two streams are fused through a set of deformable temporal residual modules (DTRMs). Best viewed in color. 43

4.2 Deep architectures of recent work: (a) Encoder-decoder temporal convolutional networks (ED-TCNs) [68], (b) Temporal convolutional U-networks (TUNets) [98], (c) Temporal residual networks (TResNets) [44]. A comparison of these architectures with our TDRN shown in Figure 4.1 makes our differences obvious: none of these models use deformable temporal convolutions and two processing streams. Best viewed in color. 44

4.3 Key differences between: (a) Common temporal residual module with a single input and output; and (b) Our deformable temporal residual module (DTRM) with two inputs and two outputs. The input represents feature sequences $f^{L-1}_{1:T/2^n}$ and $f^{L-1}_{1:T}$ with temporal lengths of $T/2^n$ and T, which are computed by the previous layer $L-1$. In (a), the output features are computed by a standard temporal convolution, $F(f^{L-1}_{1:T/2^n}; W^L)$. In (b), the output is computed using a cascade of a deformable temporal convolution, $G(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L)$, followed by a convolution and unpooling, $F(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L)$. 46

4.4 DTRM: Given the residual and pooling features of the previous layer as inputs, DTRM applies pooling to the residual stream and then concatenates the result with the input pooling stream. The concatenated features are then processed by the deformable temporal convolution, resulting in the output pooling features, $f^L_{1:T/2^n}$. Also, a temporal residual is computed from $f^L_{1:T/2^n}$ by the 1×1 temporal convolution and temporal unpooling, resulting in output residual features, $f^L_{1:T}$. 48

4.5 An illustration of deformable temporal convolution with kernel size 3 and dilation size 1. Both the temporal offsets and output features are obtained by applying a temporal convolutional layer over the same input feature maps. The offset fields have the same size as the input feature map. . . . 49

LIST OF FIGURES (Continued)

Figure Page

4.6 Action segmentations for a sample test video named *rgb-22-2.avi* from the 50 Salads dataset. Top-down, the rows correspond to ground truth sequence of actions {place lettuce into bowl, cut cheese, place cheese into bowl, peel cucumber, background, cut cucumber, place cucumber into bowl, mix ingredients, serve salad onto plate, add dressing}, and predictions of TDRN, TRN, ED-TCN [65], ST-CNN [66], Bi-LSTM [106] and Dilated TCN [65]. 51

4.7 Action segmentations for a sample test video named *S3-CofHoney-C1.mp4* from the GTEA dataset. Top-down, the rows correspond to ground truth sequence of actions { background, take, background, take, open, background, scoop, pour, background, scoop, pour, background, put, close, background, take, background, open, background, pour, background, put, close, background, take, background, open, background, pour, put, close, background, stir }, and predictions of TDRN, TRN, ED-TCN [65], Bi-LSTM [106], ST-CNN [66] and Dilated TCN [65]. 54

4.8 Action segmentations for a sample test video named *Suturing-B002.avi* from the JIGSAWS dataset. Top-down, the rows correspond to ground truth sequence of actions in different gestures {G1, G5, G8, G2, G3, G6, G4, G2, G3, G6, G4, G2, G3, G6, G4, G2, G3, G6, G11}, and predictions of TDRN, TRN, Bi-LSTM [106], ED-TCN [65], ST-CNN [66] and Dilated TCN [65]. 55

4.9 F1@10 of TDRN as a fucntion of temporal kernel size and network depth on 50Salads (mid). 57

LIST OF TABLES

Table Page

2.1 Superpixel accuracy on Youtube Face Database [125]. Error reduction in overall superpixel accuracy is cacluated w.r.t the CRF. The mean and the standard derivation are given from a 5-fold cross-validation. 17

2.2 Pixel accuracy on Cambridge-driving Labeled Video Database [9]. 19

3.1 The configuration of the decoder in FCSN. 29

3.2 Results on VSB100. 35

3.3 Results on VSB100 with fine-tuning on both BSDS500 and VSB100 training sets. 36

3.4 Quantitative results of boundary flow on Sintel training dataset in EPE metric. 36

3.5 Quantitative results on Sintel final test set. 39

4.1 Performance comparison with respect to the most related temporal convolution models including ED-TCN [65], TUNet [98] and TResNet [44]. . 50

4.2 TDRN architecture: The temporal convolution kernel is described in the same format as in Keras [17], i.e., Conv1D(filters, kernel size, strides, dilation rate). The last argument of a DTRM kernel specifies the temporal convolution kernel corresponding to offsets. C denotes the number of action classes including background class. The fully-connected layer, Dense, is applied to every temporal window of the input video. 50

4.3 Results on 50 Salads (mid). 53

4.4 Results on GTEA. 55

4.5 Results on JIGSAWS. 56

LIST OF ALGORITHMS

Algorithm Page

1 Joint Training of RTDF 15

Chapter 1: Introduction

Video labeling is a basic vision problem, and of great importance to a wide range of applications, including object detection (e.g., [32]), object proposals (e.g., [141]), video segmentation (e.g., [70, 126, 101]), depth prediction (e.g., [1]), object tracking (e.g., [75, 15, 72]), video surveillance (e.g., [112]) and robot navigation (e.g., [89]).

Video labeling can be formulated at varying levels of granularity, including (i) classifying an entire video with one label (i.e., video classification/recognition [113, 58, 122, 120, 104, 2, 135, 19, 30, 31, 38, 123]), (ii) classifying video frames with appropriate classes (i.e., frame-wise video labeling [131, 106, 27, 29, 28, 96, 62, 65]) and (iii) labeling every pixel in a video with a label (i.e., pixel-wise video labeling [55, 126, 47, 16]).

This dissertation focuses on the problem of video labeling at both the frame and pixel levels. The challenges of the problems can be summarized as follows:

- We are working with real video data. The videos are recorded in uncontrolled environments with large variations in lighting conditions and camera viewpoints. Also, objects occurring in these videos exhibit a wide variability in appearance, shape, and motion patterns, and are subject to long-term occlusions.
- We are working with big data. The large size and high dimensionality of data introduce unique computational challenges, which require specific software engineering and parallel computing techniques for efficiency.

In order to address these challenges, our research is aimed at building effective deep neural networks. Traditional approaches (e.g., [34, 40, 53, 133, 138, 80, 81, 129, 55]) typically resort to extracting hand-designed video features, and compute compatibility terms only over local space and/or time neighborhoods. In contrast, we jointly learn the feature extractor and the labeler that accounts for both local and long-range dependencies end-to-end, in a unified manner. Different from existing deep learning approaches to frame-wise labeling (e.g., [65, 67, 68]) which compute regular temporal convolutions in a single processing stream for capturing long-range video information at different scales, we compute two parallel temporal streams facilitating both local, fine-scale cues and multiscale context for improving accuracy of frame classification.

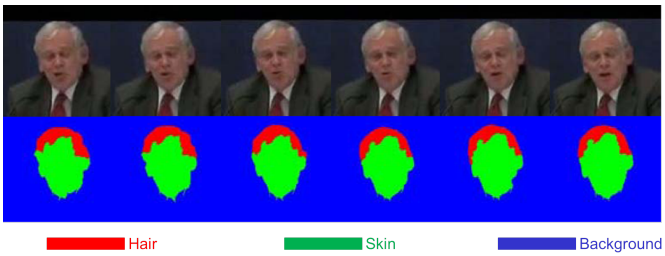


Figure 1.1: Gien a video, assign an object class label to every pixel in each video frame.

Our design of deep architectures for video labeling has been driven by the following two hypotheses: (i) Robust and accurate video labeling requires accounting for both local and long-range spatial and temporal cues; and (ii) Accuracy of pixel-level video labeling improves by modeling and inferring of object boundaries, and similarly for the frame-level video labeling, temporal action boundaries.

In this dissertation, we validate our two hypotheses on benchmark datasets. Our results were published at peer-reviewed conferences, and at the time our approaches outperformed the existing work.

In the following, we first summarize our work on pixel-wise video labeling including: (i) Recurrent temporal deep field (RTDF) [73] for spatiotemporal semantic segmentation; and (ii) Fully convolutional Siamese network (FCSN) [71] for joint boundary detection and boundary flow estimation. Then, we summarize our work on frame-wise video labeling that uses a temporal deformable residual network (TDRN) [74] for temporal action segmentation.

We have developed a recurrent temporal deep field (RTDF) [73] for semantic pixel labeling in videos. Our goal here is to assign a semantic label (e.g., bike, road, sidewalk, hair, skin) to every pixel in a video which shows natural driving scenes, captured by a camera installed on a moving car facing forward, or indoor close-ups of a persons head facing the camera, as shown in Figure 1.1. RTDF is a conditional random field (CRF) [64] that combines a deconvolution neural network (DeconvNet) [4] and a recurrent temporal restricted Boltzmann machine (RTRBM) [114], which can be jointly trained end-to-end. Our key idea has been to combine CRF and RTRBM to build a state-of-the-art video labeler, since CRF is suitable for modeling local interactions while RTRBM is appropriate for modeling global properties of object shapes. We have derived a mean-field inference

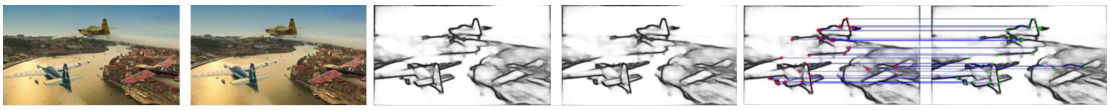


Figure 1.2: Given two consecutive video frames, detect object boundaries and estimate motions of object boundaries.

algorithm to jointly predict all latent variables in both RTRBM and CRF, and conducted end-to-end joint training of all DeconvNet, RTRBM, and CRF parameters. The joint learning and inference integrate the three components into a unified deep model RTDF. We have epirically shown on the benchmark Youtube Face Database (YFDB) [125] and Cambridge-driving Labeled Video Database (Camvid) [9] that RTDF outperforms the state of the art both qualitatively and quantitatively.

Our work on pixel labeling has addressed boundary flow estimation using a fully convolutional Siamese network (FCSN) [71]. Our goal here is joint object boundary detection and boundary motion estimation in videos, which we named boundary flow estimation (see Figure 1.2). Boundary flow is an important mid-level visual cue as boundaries characterize objects spatial extents, and the flow indicates objects motions and interactions. Yet, most prior work on motion estimation has focused on dense object motion or feature points that may not necessarily reside on boundaries. Our FCSN encodes two consecutive video frames into a coarse joint feature representation and estimates boundaries in each of the two input images via deconvolution and un-max-pooling operations. The correspondence of those predicted boundary points are estimated by decoder bridging, which uses the joint feature representation as a bridge to connect the two decoder branches. Evaluation has been conducted on three tasks: boundary detection in videos, boundary flow estimation, and optical flow estimation. We have achieved the state-of-the-art performance on boundary detection on the benchmark VSB100 dataset [33]. Besides, we have presented the first results on boundary flow estimation on the Sintel training dataset [12]. For optical flow estimation, we have augmented coarse-to-fine path match (CPM) [49] input with our boundary-flow matches, and achieved significant performance improvement on the Sintel benchmark [12].

Finally, we have specified a temporal deformable residual network (TDRN) [74] using two parallel temporal computational processes for frame labeling. As shown in Figure 1.3,

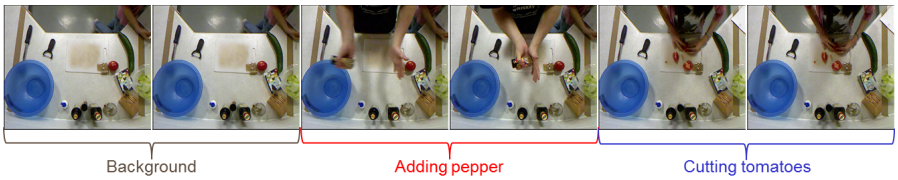


Figure 1.3: Given a video, label video frames with appropriate action classes.

the goal here is to label video frames with appropriate action classes. We have introduced a new model – temporal deformable residual network (TDRN) – aimed at analyzing video intervals at multiple temporal scales for labeling video frames. Our TDRN computes two parallel temporal processes: i) Residual stream that analyzes video information at its full temporal resolution, and ii) Pooling/unpooling stream that captures long-range video information at different scales. The former facilitates local, fine-scale action segmentation, and the latter uses multiscale context for improving accuracy of frame classification. These two streams are computed by a set of temporal residual modules with deformable convolutions, and fused by temporal residuals at the full video resolution. Our evaluation on the University of Dundee 50 Salads [110], Georgia Tech Egocentric Activities [29], and JHU-ISI Gesture and Skill Assessment Working Set [36] has demonstrated that TDRN outperforms the state of the art in frame-wise segmentation accuracy, segmental edit score, and segmental overlap F1 score.

This is a manuscript type of Ph.D. dissertation, and the following chapters are taken from our peer-reviewed publications [73, 71, 74] and accommodated for the dissertation formatting. This dissertation is organized as follows. Chapter 2 describes our recurrent temporal deep field (RTDF) for pixel labeling in videos, Chapter 3 presents our fully convolutional Siamese network (FCSN) for boundary points labeling and boundary flow estimation, Chapter 4 specifies our temporal deformable residual network (TDRN) for temporal action segmentation in videos, and Chapter 5 presents our concluding remarks.

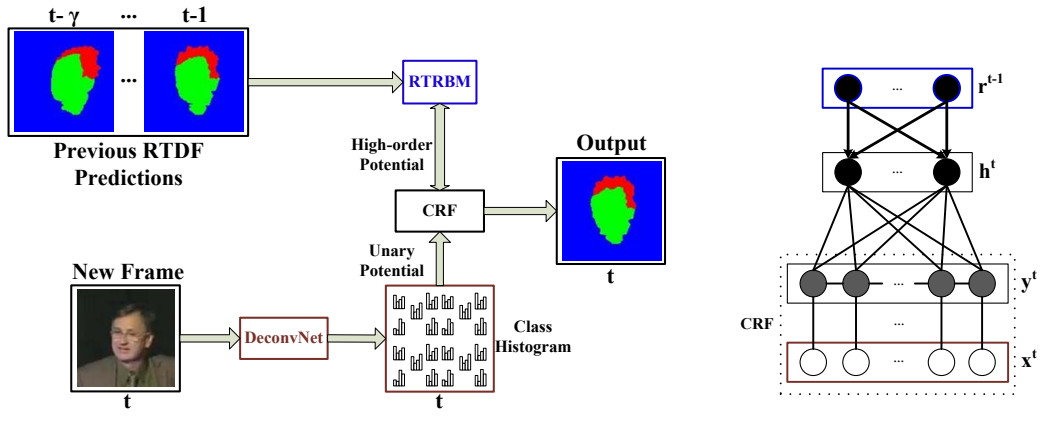
Chapter 2: Recurrent Temporal Deep Field for Semantic Video Labeling

2.1 Introduction

This chapter presents a new deep architecture for semantic video labeling, where the goal is to assign an object class label to every pixel. Our videos show natural driving scenes, captured by a camera installed on a moving car facing forward, or indoor close-ups of a person's head facing the camera. Both outdoor and indoor videos are recorded in uncontrolled environments with large variations in lighting conditions and camera viewpoints. Also, objects occurring in these videos exhibit a wide variability in appearance, shape, and motion patterns, and are subject to long-term occlusions. To address these challenges, our key idea is to efficiently account for both local and long-range spatiotemporal cues using deep learning.

Our deep architecture, called Recurrent Temporal Deep Field (RTDF), leverages the conditional random field (CRF) [64] for integrating local and contextual visual cues toward semantic pixel labeling, as illustrated in Figure 2.1. The energy of RTDF is defined in terms of unary, pairwise, and higher-order potentials.

As the unary potential, we use class predictions of the Deconvolution Neural Network (DeconvNet) [4] for every pixel of a new frame at time t. DeconvNet efficiently computes the unary potential in a feed-forward manner, through a sequence of convolutional and deconvolutional processing of pixels in frame t. Since the unary potential is computed based only on a single video frame, DeconvNet can be viewed as providing local spatial cues to our RTDF. As the pairwise potential, we use the standard spatial smoothness of pixel labels. Finally, as the higher-order potential, we use hidden variables of the Recurrent Temporal Restricted Boltzmann Machine (RTRBM) [114] (see Figure 2.1b). This hidden layer of RTRBM is computed from a sequence of previous RTDF predictions for pixels in frames $\{t-1, t-2, \ldots, t-\gamma\}$. RTRBM is aimed at capturing long-range spatiotemporal dependencies among already predicted pixel labels, which is then used to enforce spatiotemporal coherency of pixel labeling in frame t.



(a) (b)

Figure 2.1: (a) Our semantic labeling for a Youtube Face video [125] using RTDF. Given a frame at time t, RTDF uses a CRF to fuse both local and long-range spatiotemporal cues for labeling pixels in frame t. The local cues (red box) are extracted by DeconvNet [4] using only pixels of frame t. The long-range spatiotemporal cues (blue box) are estimated by RTRBM [114] (precisely, the hidden layer of RTDF) using a sequence of previous RTDF predictions for pixels in frames $t-1, t-2, \ldots, t-\gamma$. (b) An illustration of RTDF with pixel labels $\mathbf{y}^t$ in frame t, unary potentials $\mathbf{x}^t$, and top two layers $\mathbf{r}^{t-1}$ and $\mathbf{h}^t$ belonging to RTRBM. The high-order potential is distributed to all pixels in frame t via the full connectivity of layers $\mathbf{h}^t$ and $\mathbf{y}^t$, and layers $\mathbf{r}^{t-1}$ and $\mathbf{h}^t$.

We formulate a new mean-field inference algorithm to jointly predict all latent variables in both RTRBM and CRF. We also specify a joint end-to-end learning of CRF, DeconvNet and RTRBM. The joint learning and inference integrate the three components into a unified deep model – RTDF.

The goal of inference is to minimize RTDF energy. Input to RTDF inference at frame t consists of: (a) pixels of frame t, and (b) RTDF predictions for pixels in frames $\{t-1, \ldots, t-\gamma\}$. Given this input, our mean-field inference algorithm *jointly* predicts hidden variables of RTRBM and pixel labels in frame t.

Parameters of CRF, DeconvNet, and RTRBM are *jointly* learned in an end-to-end fashion, which improves our performance over the case when each component of RTDF is independently trained (a.k.a. piece-wise trained).

Our semantic video labeling proceeds frame-by-frame until all frames are labeled. Note that for a few initial frames $t \leq \gamma$, we do not use the high-order potential, but only the unary and pairwise potentials in RTDF inference.

Our contributions are summarized as follows:

1. A new deep architecture, RTDF, capable of efficiently capturing both local and long-range spatiotemporal cues for pixel labeling in video,

2. An efficient mean-field inference algorithm that jointly predicts hidden variables in RTRBM and CRF and labels pixels; as our experiments demonstrate, our mean-field inference yields better accuracy of pixel labeling than an alternative stage-wise inference of each component of RTDF.

3. A new end-to-end joint training of all components of RTDF using loss backpropagation; as our experiments demonstrate, our joint training outperforms the case when each component of RTDF is trained separately.

4. Improved pixel labeling accuracy relative to the state of the art, under comparable runtimes, on the benchmark datasets.

In the following, Sec. 2.2 reviews closely related work; Sec. 2.3 specifies RTDF and briefly reviews its basic components: RBM in Sec. 2.3.1, RTRBM in Sec. 2.3.2, and DeconvNet in Sec. 2.3.3; Sec. 2.4 formulates RTDF inference; Sec. 2.5 presents our training of RTDF; and Sec. 2.6 shows our experimental results.

2.2 Related Work

This section reviews closely related work on semantic video labeling, whereas the literature on unsupervised and semi-supervised video segmentation is beyond our scope. We also discuss our relationship to other related work on semantic image segmentation, and object shape modeling.

Semantic video labeling has been traditionally addressed using hierarchical graphical models (e.g., [34, 40, 53, 133, 138, 80]). However, they typically resort to extracting hand-designed video features for capturing context, and compute compatibility terms only over local space-time neighborhoods.

Our RTDF is related to semantic image segmentation using CNNs [26, 82, 13, 18, 92, 42, 41, 35, 140]. These approaches typically use multiple stages of training, or iterative component-wise training. Instead, we use a joint training of all components of our deep architecture. For example, a fully convolutional network (FCN) [82] is trained in a stage-wise manner such that a new convolution layer is progressively added to a previously trained network until no performance improvement is obtained. For smoothness, DeepLab [13] uses a fully-connected CRF to post-process CNN predictions, while the CRF and CNN are iteratively trained, one at a time. Also, a deep deconvolution network presented in [88] uses object proposals as a pre-processing step. For efficiency, we instead use DeconvNet [4], as the number of trainable parameters in DeconvNet is significantly smaller in comparison to peer deep networks.

RTDF is also related to prior work on restricted Boltzmann machine (RBM) [109]. For example, RBMs have been used for extracting both local and global features of object shapes [45], and shape Boltzmann machine (SBM) can generate deformable object shapes [25]. Also, RBM has been used to provide a higher-order potential for a CRF in scene labeling [78, 56].

The most related model to ours is the shape-time random field (STRF) [55]. STRF combines a CRF with a conditional restricted Boltzmann machine (CRBM) [116] for video labeling. They use CRBM to estimate a higher-order potential of the STRF's energy. While this facilitates modeling long-range shape and motion patterns of objects, input to their CRF consists of hand-designed features. Also, they train CRF and CRBM iteratively, as separate modules, in a piece-wise manner. In contrast, we jointly learn all components of our RTDF in a unified manner via loss backpropagation.

2.3 Recurrent Temporal Deep Field

Our RTDF is an energy-based model that consists of three components – DeconvNet, CRF, and RTRBM – providing the unary, pairwise, and high-order potentials for predicting class labels $\mathbf{y}^t = \{\mathbf{y}_p^t : \mathbf{y}_p^t \in \{0,1\}^L\}$ for pixels p in video frame t, where $\mathbf{y}_p^t$ has only one non-zero element. Labels $\mathbf{y}^t$ are predicted given: (a) pixels $\mathbf{I}^t$ of frame t, and (b) previous RTDF predictions $\mathbf{y}^{<t} = \{\mathbf{y}^{t-1}, \mathbf{y}^{t-2}, \ldots, \mathbf{y}^{t-\gamma}\}$, as illustrated in Figure 2.2.

DeconvNet takes pixels $\mathbf{I}^t$ as input, and outputs the class likelihoods $\mathbf{x}^t = \{\mathbf{x}_p^t : \mathbf{x}_p^t \in [0,1]^L, \sum_{l=1}^{L} x_{pl}^t = 1\}$, for every pixel p in frame t. A more detailed description of

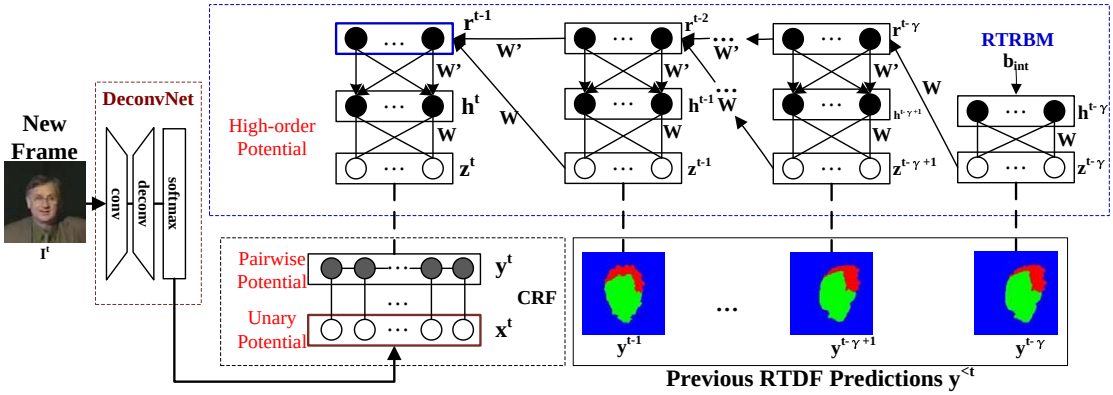


Figure 2.2: Our RTDF is an energy-based model that predicts pixel labels $\mathbf{y}^t$ for frame t, given the unary potential $\mathbf{x}^t$ of DeconvNet, the pairwise potential between neighboring pixel labels in frame t, and the high-order potential defined in terms of $\mathbf{z}^t$, $\mathbf{h}^t$ and $\mathbf{r}^{t-1}$ of RTRBM. The figure shows the time-unfolded visualization of computational processes in RTRBM. RTRBM takes as input previous RTDF predictions $\{\mathbf{y}^{t-1}, \ldots, \mathbf{y}^{t-\gamma}\}$ and encodes the long-range and high-order dependencies through latent variables $\mathbf{r}^{t-1}$. The high-order potential is further distributed to all pixels in frame t via a deterministic mapping (vertical dashed lines) between $\mathbf{y}^t$ and $\mathbf{z}^t$.

DeconvNet is given in Sec. 2.3.3. $\mathbf{x}^t$ is then used to define the unary potential of RTDF.

RTRBM takes previous RTDF predictions $\mathbf{y}^{<t}$ as input and estimates values of latent variables $\mathbf{r}^{<t} = \{\mathbf{r}^{t-1}, \ldots, \mathbf{r}^{t-\gamma}\}$ from $\mathbf{y}^{<t}$. The time-unfolded visualization in Figure 2.2 shows that $\mathbf{r}^{t-1}$ is affected by previous RTDF predictions $\mathbf{y}^{<t}$ through the full connectivity between two consecutive $\mathbf{r}$ layers and the full connectivity between the corresponding $\mathbf{r}$ and $\mathbf{z}$ layers.

The hidden layer $\mathbf{r}^{t-1}$ is aimed at capturing long-range spatiotemporal dependences of predicted class labels in $\mathbf{y}^{<t}$. Thus, $\mathbf{h}^t$ and $\mathbf{r}^{t-1}$ are used to define the high-order potential of RTDF, which is distributed to all pixels in frame t via the full connectivity between layers $\mathbf{h}^t$ and $\mathbf{z}^t$, as well as between layers $\mathbf{h}^t$ and $\mathbf{r}^{t-1}$ in RTRBM. Specifically, the high-order potential is distributed to each pixel via a deterministic mapping between nodes in $\mathbf{z}^t$ and pixels in $\mathbf{y}^t$. While there are many options for this mapping, in our implementation, we partition frame t into a regular grid of patches. As further explained in Sec. 2.3.1, each node of $\mathbf{z}^t$ is assigned to a corresponding patch of pixels in $\mathbf{y}^t$.

The energy of RTDF is defined as

$$E_{\text{RTDF}}(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t) = -\sum_p \psi_1(\mathbf{x}_p^t, \mathbf{y}_p^t) - \sum_{p,p'} \psi_2(\mathbf{y}_p^t, \mathbf{y}_{p'}^t) + E_{\text{RT}}(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}). \tag{2.1}$$

In (2.1), the first two terms denote the unary and pairwise potentials, and the third term represents the high-order potential. As mentioned above, the mapping between $\mathbf{y}^t$ and $\mathbf{z}^t$ is deterministic. Therefore, instead of using $\mathbf{z}^t$ in (2.1), we can specify E_{RT} directly in terms of $\mathbf{y}^t$. This allows us to conduct *joint* inference of $\mathbf{y}^t$ and $\mathbf{h}^t$, as further explained in Sec. 2.4.

The unary and pairwise potentials are defined as for standard CRFs:

$$\psi_1(\mathbf{x}_p^t, \mathbf{y}_p^t) = W^1_{\mathbf{y}_p^t} \cdot \mathbf{x}_p^t, \qquad \psi_2(\mathbf{y}_p^t, \mathbf{y}_{p'}^t) = W^2_{\mathbf{y}_p^t, \mathbf{y}_{p'}^t} \cdot \exp(-|\mathbf{x}_p^t - \mathbf{x}_{p'}^t|), \tag{2.2}$$

where $W^1_{\mathbf{y}} \in \mathbb{R}^L$ is an L-dimensional vector of unary weights for a given class label at pixel p, and $W^2_{\mathbf{y},\mathbf{y}'} \in \mathbb{R}^L$ is an L-dimensional vector of pairwise weights for a given pair of class labels at neighboring pixels p and p'.

Before specifying E_{RT}, for clarity, we first review the restricted Boltzmann machine (RBM) and then explain its extension to RTRBM.

2.3.1 A Brief Review of Restricted Boltzmann Machine

RTRBM can be viewed as a temporal concatenation of RBMs [114]. RBM [109] is an undirected graphical model with one visible layer and one hidden layer. In our approach, the visible layer consists of L-dimensional binary vectors $\mathbf{z} = \{\mathbf{z}_i : \mathbf{z}_i \in \{0,1\}^L\}$ and each $\mathbf{z}_i$ has only one non-zero element representing the class label of the corresponding patch i in a given video frame. The hidden layer consists of binary variables $\mathbf{h} = \{h_j : h_j \in \{0,1\}\}$. RBM defines a joint distribution of the visible layer $\mathbf{z}$ and the hidden layer $\mathbf{h}$, and the energy function between the two layers for a video frame is defined as:

$$E_{\text{RBM}}(\mathbf{z}, \mathbf{h}) = -\sum_j \sum_i \sum_{l=1}^{L} W_{ijl} h_j z_{il} - \sum_i \sum_{l=1}^{L} z_{il} c_{il} - \sum_j b_j h_j \tag{2.3}$$

where W is the RBM's weight matrix between $\mathbf{z}$ and $\mathbf{h}$, and $\mathbf{b}$ and $\mathbf{c}$ are the bias vectors for $\mathbf{h}$ and $\mathbf{z}$, respectively. RBM has been successfully used for modeling spatial context of an image or video frame [78, 56, 55].

Importantly, to reduce the huge number of parameters in RBM (and thus facilitate learning), we follow the pooling approach presented in [55]. Specifically, instead of working directly with pixels in a video frame, our formulation of RBM uses patches i of pixels (8×8 pixels) as corresponding to the visible variables $\mathbf{z}_i$. The patches are obtained by partitioning the frame into a regular grid.

Recall that in our overall RTDF architecture RBM is grounded onto latent pixel labels $\mathbf{y}_p$ through the deterministic mapping of $\mathbf{z}_i$'s to pixels p that fall within patches i (see Figure 2.2). When predicted labels $\mathbf{y}^{<t}$ are available for video frames before time t, we use the following mapping $\mathbf{z}_i = 1/|i| \sum_{p \in i} \mathbf{y}_p$, where $|i|$ denotes the number of pixels in patch i. Note that this will give real-valued $\mathbf{z}_i$'s, which we then binarize. Conversely, for frame t, when we want to distribute the high-order potential, we deterministically assign potential of $\mathbf{z}_i$ to every pixel within the patch.

2.3.2 A Brief Review of RTRBM

RTRBM represents a recurrent temporal extension of an RBM [114], with one visible layer $\mathbf{z}$, and two hidden layers $\mathbf{h}$ and $\mathbf{r}$. As in RBM, $\mathbf{h}$ are binary variables, and $\mathbf{r} = \{r_j : r_j \in [0,1]\}$ represents a set of real-valued hidden variables. In the time-unfolded

visualization shown in Figure 2.2, RTRBM can be seen as a temporal concatenation of the respective sets of RBM's variables, indexed by time t, $\{\mathbf{z}^t, \mathbf{h}^t, \mathbf{r}^t\}$. This means that each RBM at time t in RTRBM has a dynamic bias input that is affected by the RBMs of previous time instances. This dynamic bias input is formalized as a recurrent neural network [100], where hidden variables $\mathbf{r}^t$ at time t are obtained as

$$\mathbf{r}^t = \sigma(W\mathbf{z}^t + \mathbf{b} + W'\mathbf{r}^{t-1}), \tag{2.4}$$

where $\{\mathbf{b}, W, W'\}$ are parameters. Note that $\mathbf{b}+W'\mathbf{r}^{t-1}$ is replaced by $\mathbf{b}_{int}$ for time $t = 1$, $\sigma(\cdot)$ is the element-wise sigmoid function, and W' is the shared weight matrix between $\mathbf{r}^{t-1}$ and $\mathbf{h}^t$ and between $\mathbf{r}^{t-1}$ and $\mathbf{r}^t$. Consequently, the recurrent neural network in RTRBM is designed such that the conditional expectation of $\mathbf{h}^t$, given $\mathbf{z}^t$, is equal to $\mathbf{r}^t$. RTRBM defines an energy of $\mathbf{z}^t$ and $\mathbf{h}^t$ conditioned on the hidden recurrent input $\mathbf{r}^{t-1}$ as

$$E_{\mathrm{RT}}(\mathbf{z}^t, \mathbf{h}^t|\mathbf{r}^{t-1}) = E_{\mathrm{RBM}}(\mathbf{z}^t, \mathbf{h}^t) - \sum_j \sum_k W'_{jk} h_j^t r_k^{t-1}. \tag{2.5}$$

From (2.3), (2.4) and (2.5), RTRBM parameters are $\theta_{\mathrm{RT}} = \{\mathbf{b}_{\mathrm{int}}, \mathbf{b}, \mathbf{c}, W, W'\}$. The associated free energy of $\mathbf{z}^t$ is defined as

$$F_{\mathrm{RT}}(\mathbf{z}^t|\mathbf{r}^{t-1}) = -\sum_j \log(1 + \exp(b_j + \sum_{i,l} W_{ijl} z_{il} + \sum_k W'_{jk} r_k^{t-1})) - \sum_{i,l} z_{il} c_{il}. \tag{2.6}$$

RTRBM can be viewed as capturing long-range and high-order dependencies in both space and time, because it is characterized by the full connectivity between consecutive $\mathbf{r}$ layers, and between the corresponding $\mathbf{r}$, $\mathbf{z}$, and $\mathbf{h}$ layers.

Due to the deterministic mapping between $\mathbf{z}^t$ and $\mathbf{y}^t$ for frame t, we can specify E_{RT} given by (2.5) in terms of $\mathbf{y}^t$, i.e., as $E_{\mathrm{RT}}(\mathbf{y}^t, \mathbf{h}^t|\mathbf{r}^{t-1})$. We will use this to derive a mean-field inference of $\mathbf{y}^t$, as explained in Sec. 2.4.

2.3.3 DeconvNet

As shown in Figure 2.2, DeconvNet [4] is used for computing the unary potential of RTDF. We strictly follow the implementation presented in [4]. DeconvNet consists of

two networks: one based on VGG16 net to encode the input video frame, and a multilayer deconvolution network to generate feature maps for predicting pixel labels. The convolution network records the pooling indices computed in the pooling layers. Given the output of the convolution network and the pooling indices, the deconvolution network performs a series of unpooling and deconvolution operations for producing the final feature maps. These feature maps are passed through the softmax layer for predicting the likelihoods of class labels of every pixel, $\mathbf{x}_p \in [0,1]^L$. Before joint training, we pre-train parameters of DeconvNet, θ_{DN}, using the cross entropy loss, as in [4].

2.4 Inference of RTDF

Pixel labels of the first γ frames of a video are *predicted* using a variant of our model – namely, the jointly trained CRF + DeconvNet, without RTRBM. Then, inference of the full RTDF (i.e., jointly trained CRF + DeconvNet + RTRBM) proceeds to subsequent frames until all the frames have been labeled.

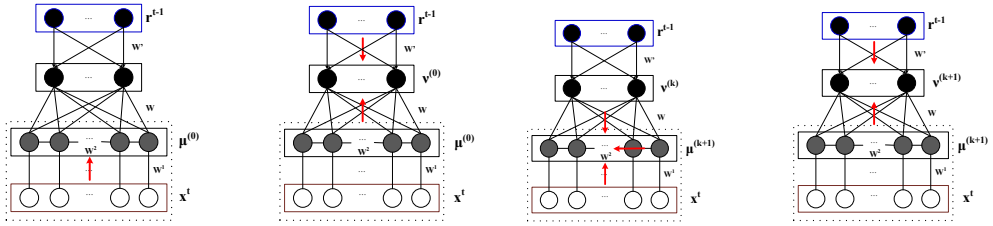
Given a sequence of semantic labelings in the past, $\mathbf{y}^{<t}$, and a new video frame, $\mathbf{I}^t$, the goal of RTDF inference is to predict $\mathbf{y}^t$ as:

$$\hat{\mathbf{y}}^t = \arg\max_{\mathbf{y}^t} \sum_{\mathbf{h}^t} \exp(-E_{\text{RTDF}}(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t)). \tag{2.7}$$

Since the exact inference of RTDF is intractable, we formulate an approximate mean-field inference for jointly predicting both $\hat{\mathbf{y}}^t$ and $\hat{\mathbf{h}}^t$. Its goal is to minimize the KL-divergence between the true posterior distribution, $P(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t) = \frac{1}{Z(\theta)} \exp(-E_{\text{RTDF}}(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t))$, and the mean-field distribution $Q(\mathbf{y}^t, \mathbf{h}^t) = \prod_p Q(\mathbf{y}_p^t) \prod_j Q(\mathbf{h}_j^t)$ factorized over pixels p for $\mathbf{y}^t$ and hidden nodes j for $\mathbf{h}^t$.

To derive our mean-field inference, we introduce the following two types of variational parameters: (i) $\boldsymbol{\mu} = \{\mu_{pl} : \mu_{pl} = Q(\mathbf{y}_{pl}^t = 1)\}$, where $\sum_{l=1}^L \mu_{pl} = 1$ for every pixel p; (ii) $\boldsymbol{\nu} = \{\nu_j : \nu_j = Q(\mathbf{h}_j^t = 1)\}$. They allow us to express the mean-field distribution as $Q(\mathbf{y}^t, \mathbf{h}^t) = Q(\boldsymbol{\mu}, \boldsymbol{\nu}) = \prod_p \mu_p \prod_j \nu_j$. It is straightforward to show that minimizing the KL-divergence between P and Q amounts to the following objective

$$\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}} = \arg\max_{\boldsymbol{\mu},\boldsymbol{\nu}} \{ \sum_{\mathbf{y}^t, \mathbf{h}^t} Q(\boldsymbol{\mu}, \boldsymbol{\nu}) \ln P(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t) + H(Q(\boldsymbol{\mu}, \boldsymbol{\nu})) \}, \tag{2.8}$$



(a) (b) (c) (d)

Figure 2.3: Key steps of the mean-field inference overlaid over RTDF which is depicted as in Figure 2.1b. (a) Initialization of $\boldsymbol{\mu}^{(0)}$. (b) Initialization of $\boldsymbol{\nu}^{(0)}$. (c) Updating of $\boldsymbol{\mu}^{(k+1)}$. (d) Updating of $\boldsymbol{\nu}^{(k+1)}$. The red arrows show the information flow.

where $H(Q)$ is the entropy of Q.

Our mean-field inference begins with initialization: $\mu_{pl}^{(0)} = \frac{\exp(W^1_{\mu_{pl}} \cdot \mathbf{x}_p)}{\sum_{l'} \exp(W^1_{\mu_{pl'}} \cdot \mathbf{x}_p)}$, $\nu_j^{(0)} = \sigma(\sum_l \sum_i \sum_{p \in i} \frac{1}{|i|} \mu_{pl}^{(0)} W_{ijl} + b_j + \sum_{j'} W'_{jj'} r_{j'}^{t-1})$ and then proceeds by updating $\mu_{pl}^{(k)}$ and $\nu_j^{(k)}$ using the following equations until convergence:

$$\mu_{pl}^{(k+1)} = \frac{\exp(W^1_{\mu_{pl}^{(k)}} \cdot \mathbf{x}_p + \sum_j W_{ijl} \nu_j^{(k)} + c_{il} + \beta_{p' \to p}^{(k)})}{\sum_{l'} \exp(W^1_{\mu_{pl'}^{(k)}} \cdot \mathbf{x}_p + \sum_j W_{ijl'} \nu_j^{(k)} + c_{il'} + \beta_{p' \to p}^{(k)})}, \tag{2.9}$$

$$\nu_j^{(k+1)} = \sigma(\sum_l \sum_i \sum_{p \in i} \frac{1}{|i|} \mu_{pl}^{(k+1)} W_{ijl} + b_j + \sum_{j'} W'_{jj'} r_{j'}^{t-1}), \tag{2.10}$$

where $\beta_{p' \to p}^{(k)} = \sum_{p'} \sum_{l'} W^2_{\mu_{pl}^{(k)}, \mu_{p'l'}^{(k)}} \cdot \exp(-|\mathbf{x}_p - \mathbf{x}_{p'}|)$ denotes a pairwise term that accounts for all neighbors p' of p, W^1 and W^2 denote parameters of the unary and pairwise potentials defined in (2.2), and W_{ijl} and $W'_{jj'}$ are parameters of RTRBM. Also, the second and the third terms in (2.9) and the first term in (2.10) use the deterministic mapping between patches i and pixels $p \in i$

(see Sec. 2.3.1). Figure 2.3 shows the information flow in our mean-field inference, overlaid over RTDF which is depicted in a similar manner as in Figure 2.1b.

After convergence at step K, the variational parameter $\boldsymbol{\mu}^{(k)}$, $k \in \{0, 1, \cdots, K\}$ associated with minimum free energy as defined in (2.12) is used to predict the label of pixels

Algorithm 1 Joint Training of RTDF

INPUT: Training set: $\{\mathbf{I}^t, \mathbf{y}^t, t = 1, 2, \cdots\}$, where $\mathbf{y}^t$ is ground truth
OUTPUT: Parameters of RTDF

1: **repeat**
2: I. For every training video, conduct the mean-field inference, presented in Sec.2.4, and calculate the free energy associated with $\mathbf{y}^t$ using (2.12);
3: II. Compute the derivative of $\triangle(\theta)$ given by (2.11) with respect to:
4: II.a. Unary term $\mathbf{x}_p$, using (2.11) and (2.2),
5: II.b. Pairwise term $\exp(-|\mathbf{x}_p^t - \mathbf{x}_{p'}^t|)$, using (2.11) and (2.2);
6: III. Update CRF parameters W^1, W^2, using the result of Step II;
7: IV. Backpropagate the result of Step II.a. to DeconvNet using the chain rule in order to update θ_{DN};
8: V. Compute $\frac{\partial \triangle}{\partial \theta_{\mathrm{RT}}}$ using (2.11), (2.12), (2.6) and (2.4) for updating θ_{RT};
9: **until** stopping criteria

in frame t. The label at every pixel p is predicted as l for which $\mu_{pl}^{(k)}$, $l \in \{1, 2, \cdots, L\}$ is maximum. This amounts to setting $\hat{y}_{pl}^t = 1$, while all other elements of vector $\hat{\mathbf{y}}_p^t$ are set to zero. Also, the value of $\hat{h}_j^t$ is estimated by binarizing the corresponding maximum $\nu_j^{(k)}$.

2.5 Learning

Parameters of all components of RTDF, $\theta = \{W^1, W^2, \theta_{\mathrm{DN}}, \theta_{\mathrm{RT}}\}$, are trained jointly. For a suitable initialization of RTDF, we first pretrain each component, and then carry out joint training, as summarized in Alg. 1.

Pretraining. (1) **RTRBM.** The goal of learning RTRBM is to find parameters θ_{RTRBM} that maximize the joint log-likelihood, $\log p(\mathbf{z}^{<t}, \mathbf{z}^t)$. To this end, we closely follow the learning procedure presented in [114], which uses the backpropagation-through-time (BPTT) algorithm [100] for back-propagating the error of patch labeling. As in [114], we use contrastive divergence (CD) [46] to approximate the gradient in training RTRBM. (2) **DeconvNet.** As initial parameters, DeconvNet uses parameters of VGG16 network (without the fully-connected layers) for the deep convolution network, and follows the approach of [4] for the deconvolution network. Then, the two components of DeconvNet are jointly trained using the cross entropy loss defined on pixel label predictions. (3) **CRF.** The CRF is pretrained on the output features from DeconvNet using

loopy belief propagation with the LBFGS optimization method.

Joint Training of RTDF. The goal of joint training is to maximize the conditional log-likelihood $\sum_t \log p(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t)$. We use CD-PercLoss algorithm [86] and error back-propagation (EBP) to jointly train parameters of RTDF in an end-to-end fashion. The training objective is to minimize the following generalized perceptron loss [69] with regularization:

$$\triangle(\theta) = \sum_t (F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t) - \min_{\hat{\mathbf{y}}^t} F(\hat{\mathbf{y}}^t|\mathbf{y}^{<t}, \mathbf{I}^t)) + \lambda\theta^T\theta \tag{2.11}$$

where $\lambda > 0$ is a weighting parameter, and $F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t)$ denotes the free energy of ground truth label $\mathbf{y}^t$ of frame t, and $\hat{\mathbf{y}}^t$ is the predicted label associated with minimum free energy. The free energy of RTDF is defined as

$$F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t) = -\sum_p \psi_1(\mathbf{x}_p^t, \mathbf{y}_p^t) - \sum_{p,p'} \psi_2(\mathbf{y}_p^t, \mathbf{y}_{p'}^t) + F_{\mathrm{RT}}(\mathbf{y}^t|\mathbf{r}^{t-1}) \tag{2.12}$$

where the first two terms denote the unary and pairwise potentials, and the third term is defined in (2.6). In the prediction pass of training, the pixel label is obtained by the mean-field inference, as explained in Sec.2.4. In the updating phase of training, the errors are back-propagated through CRF, DeconvNet and RTRBM in a standard way, resulting in a joint update of θ.

2.6 Results

Datasets and Metrics: For evaluation, we use the Youtube Face Database (YFDB) [125] and Cambridge-driving Labeled Video Database (CamVid) [9]. Both datasets are recorded in uncontrolled environment, and present challenges in terms of occlusions, and variations of motions, shapes, and lighting. CamVid consists of four long videos showing driving scenes with various object classes, whose frequency of appearance is unbalanced. Unlike other available datasets [76, 37, 11], YFDB and CamVid provide sufficient training samples for learning RTRBM. Each YFDB video contains 49 to 889 roughly aligned face images with resolution 256×256. We use the experimental setup of [55] consisting of randomly selected 50 videos from YFDB, with ground-truth labels of hair, skin, and background provided for 11 consecutive frames per each video (i.e., 550 labeled frames),

Model	Error Redu	Overall Accu	Hair	Skin	Background	Category Avg
CRF [64]	0.0	0.90 ± 0.005	0.63 ± 0.047	0.89 ± 0.025	0.96 ± 0.005	0.83 ± 0.009
GLOC [56]	0.03 ± 0.025	0.91 ± 0.006	0.61 ± 0.038	0.90 ± 0.023	0.96 ± 0.003	0.82 ± 0.008
STRF [55]	0.12 ± 0.025	0.91 ± 0.006	0.72 ± 0.039	0.89 ± 0.025	0.96 ± 0.004	0.86 ± 0.010
RTDF†	0.11 ± 0.027	0.91 ± 0.008	0.70 ± 0.043	0.89 ± 0.024	0.96 ± 0.004	0.85 ± 0.011
RTDF*	0.17 ± 0.028	0.92 ± 0.008	0.76 ± 0.049	0.88 ± 0.025	0.96 ± 0.003	0.87 ± 0.012
RTDF	**0.34 ± 0.031**	**0.93 ± 0.010**	**0.80 ± 0.037**	**0.90 ± 0.026**	**0.97 ± 0.005**	**0.89 ± 0.014**

Table 2.1: Superpixel accuracy on Youtube Face Database [125]. Error reduction in overall superpixel accuracy is cacluated w.r.t the CRF. The mean and the standard derivation are given from a 5-fold cross-validation.

which are then split into 30, 10, and 10 videos for training, validation, and testing, respectively. Each CamVid video contains 3600 to 11000 frames at resolution 360×480. CamVid provides ground-truth pixel labels of 11 object classes for 700 frames, which are split into 367 training and 233 test frames. For fair comparison on CamVid with [4], which uses significantly more training data, we additionally labeled 9 consecutive frames preceding every annotated frame in the training set of CamVid, resulting in 3670 training frames.

For fair comparison, we evaluate our superpixel accuracy on YFDB and pixel accuracy on Camvid. For YFDB, we extract superpixels as in [55] producing 300-400 superpixels per frame. The label of a superpixel is obtained by pixel majority voting. Both overall accuracy and class-specific accuracy are computed as the number of superpixels/pixels classified correctly divided by the total number of superpixels/pixels. Evaluation is done for each RTDF prediction on a test frame after processing 3 and 4 frames preceding that test frame for YFDB and Camvid, respectively.

Implementation Details: We partition video frames using a 32×32 regular grid for YFDB, and a 60×45 regular grid for CamVid. For YFDB, we specify RTRBM with 1000 hidden nodes. For CamVid, there are 1200 hidden nodes in RTRBM. Hyper-parameters of the DeconvNet are specified as in [4]. The DeconvNet consists of: (a) Convolution network with 13 convolution layers based on VGG16 network, each followed by a batch normalization operation [52] and a RELU layer; (b) Deconvolution network with 13 deconvolution layers, each followed by the batch normalization layer and the RELU layer; and (c) Soft-max layer producing a $1 \times L$ class distribution for every pixel in the image. We test $\lambda \in [0, 1]$ on the validation set, and report our test results for λ with the best performance on the validation dataset.

Runtimes: We implement RTDF on NVIDIA Tesla K80 GPU accelerator. It takes about 23 hours to train RTDF on CamVid. The average runtime for predicting pixel labels in an image with resolution 360×480 is 105.3ms.

Baselines: We compare RTDF with its variants and related work: 1) RTDF†: RTDF without end-to-end joint training (i.e., piece wise training); 2) RTDF*: jointly trained RTDF without joint inference, i.e., using stage-wise inference where the output of RTRBM is treated as fixed input into the CRF. 3) CRF [64]: spatial CRF inputs with hand-engineered features; 4) GLOC [56]: a jointly trained model that combines spatial CRF and RBM; and 5) STRF [55]: a piece-wise trained model that combines spatial CRF, CRBM and temporal potentials between two consecutive frames.

2.6.1 Quantitative Results

YFDB: Table 2.1 presents the results of the state of the art, RTDF and its variant baselines on YFDB. As can be seen, RTDF gives the best performance, since RTDF accounts for long-range spatiotemporal dependencies and performs joint training and joint inference. It outperforms STRF [55] which uses local hand-engineered features and piece-wise training. These results suggest that accounting for object interactions across a wide range of spatiotemporal scales is critical for video labeling. We also observe that RTDF† achieves comparable results with STRF [55], while RTDF* outperforms both. This suggests that our end-to-end joint training of all components of RTDF is more critical for accurate video labeling than their joint inference. Also, as RTDF* gives an inferior performance to RTDF, performing joint instead of stage-wise inference gives an additional gain in performance. Finally, we observe that RTDF performance can be slightly increased by using a larger γ. For fair comparison, we use the same γ as in [55].

CamVid: Table 2.2 presents the results of the state of the art, RTDF and its variants on CamVid. In comparison to the state of the art and the baselines, RTDF achieves superior performance in terms of both average and weighted accuracy, where weighted accuracy accounts for the class frequency. Unlike RTDF, SegNet [4] treats the label of each pixel independently by using a soft-max classifier, and thus may poorly perform around low-contrast object boundaries. On the other hand, SegNet has an inherent bias to label larger pixel areas with a unique class label [4] (see Figure 2.4), which may explain its better performance than RTDF on the following classes: sign-symbol, column-pole,

Method	Building	Tree	Sky	Car	Sign	Road	Pedestrain	Fence	Col. Pole	Sidewalk	Bicycle	Class Avg.	Global Avg.
Dense Depth Maps [136]	85.3	57.3	95.4	69.2	46.5	**98.5**	23.8	44.3	22.0	38.1	28.7	55.4	82.1
Super Parsing [119]	87.0	67.1	96.9	62.7	30.1	95.9	14.7	17.9	1.7	70.0	19.4	51.2	83.3
High-order CRF [111]	84.5	72.6	**97.5**	72.7	34.1	95.3	34.2	45.7	8.1	77.6	28.5	59.2	83.8
CRF + Detectors [63]	81.5	76.6	96.2	78.7	40.2	93.9	43.0	47.6	14.3	81.5	33.9	62.5	83.8
Neural Decision Forests [99]	N/A											56.1	82.1
Deeplab [13]	82.7	**91.7**	89.5	76.7	33.7	90.8	41.6	35.9	17.9	82.3	45.9	62.6	84.6
CRFasRNN [140]	84.6	91.3	92.4	79.6	43.9	91.6	37.1	36.3	27.4	82.9	33.7	63.7	86.1
SegNet [4]	73.9	90.6	90.1	86.4	**69.8**	94.5	**86.8**	**67.9**	**74.0**	94.7	52.9	80.1	86.7
RTDF†	81.8	87.9	91.5	79.2	59.8	90.4	77.1	61.5	66.6	91.2	54.6	76.5	86.5
RTDF*	83.6	89.8	92.9	78.5	61.3	92.2	79.6	61.9	67.7	92.8	56.9	77.9	88.1
RTDF	**87.1**	85.2	93.7	**88.3**	64.3	94.6	84.2	64.9	68.8	**95.3**	**58.9**	**80.5**	**89.9**

Table 2.2: Pixel accuracy on Cambridge-driving Labeled Video Database [9].

pedestrian and fence. From Table 2.2, RTDF† achieves comparable performance to that of SegNet, while RTDF* outperforms RTDF†. This is in agreement with our previous observation on YFDB that joint training of all components of RTDF is more critical than their joint inference for accurate video labeling.

2.6.2 Qualitative Evaluation

Figure 2.4 illustrates our pixel-level results on frame samples of CamVid. From the figure, we can see that our model is able to produce spatial smoothness pixel labeling. Figure 2.6 shows superpixel labeling on sample video clips from YFDB. As can be seen, on both sequences, STRF [55] gives inferior video labeling than RTDF in terms of temporal coherency and spatial consistency of pixel labels. Our spatial smoothness and temporal coherency can also be seen in Figure 2.5 which shows additional RTDF results on a longer sequence of frames from a sample CamVid video.

Empirically, we find that RTDF poorly handles abrupt scale changes (e.g., dramatic camera zoom-in/zoom-out). Also, in some cases shown in Figure 2.4 and Figure 2.5, RTDF misses tiny, elongated objects like column-poles, due to our deterministic mapping between patches of a regular grid and pixels.

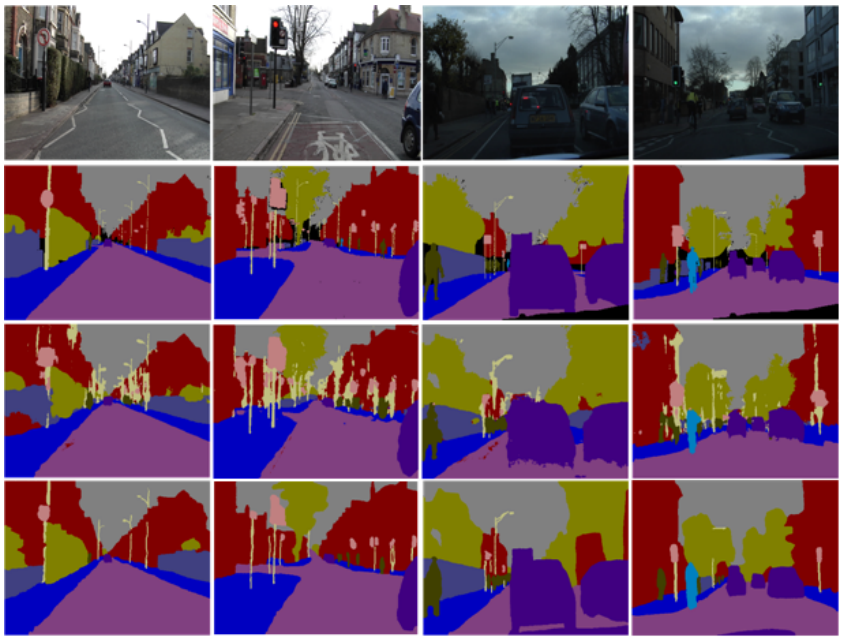


Figure 2.4: Frame samples from CamVid. The rows correspond to original images, ground truth, SegNet [4], and RTDF.

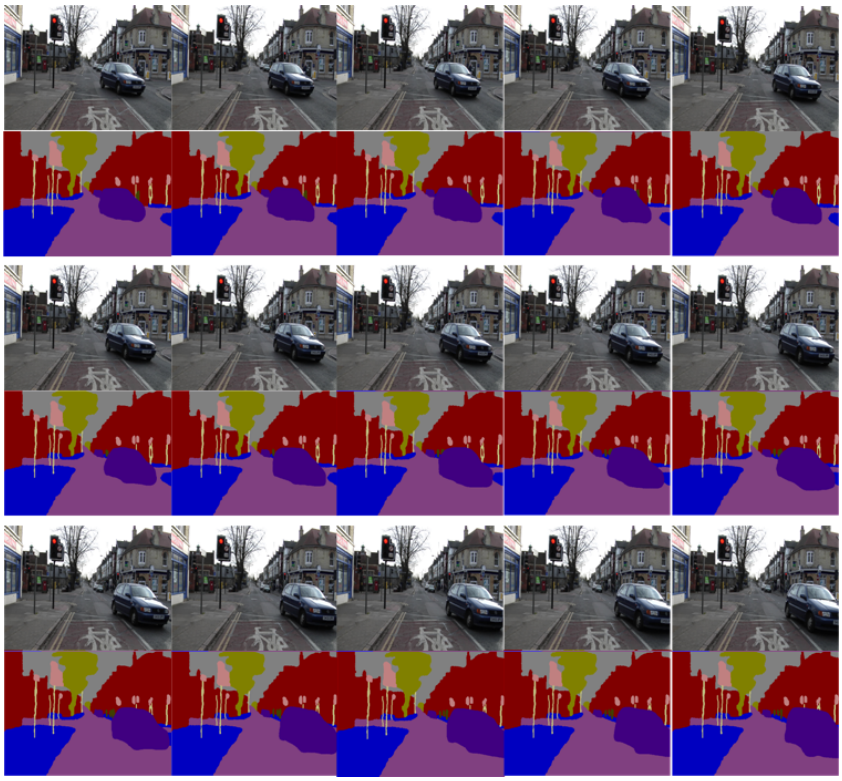


Figure 2.5: Sequence of frames from a sample CamVid video. The rows correspond to input frames and RTDF outputs.

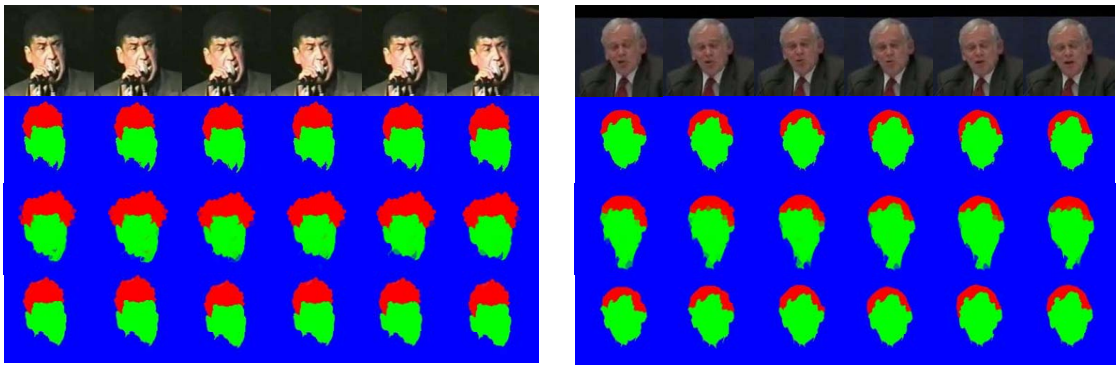


Figure 2.6: Frame sequences from two CamVid video clips. The rows correspond to original video frames, ground truth, STRF[55], and RTDF.

2.7 Summary

We have presented a new deep architecture, called Recurrent-Temporal Deep Field (RTDF), for semantic video labeling. RTDF captures long-range and high-order spatiotemporal dependencies of pixel labels in a video by combining conditional random field (CRF), deconvolution neural network (DeconvNet), and recurrent temporal restricted Boltzmann machine (RTRBM) into a unified framework. Specifically, we have derived a mean-field inference algorithm for jointly predicting latent variables in both CRF and RTRBM, and specified an end-to-end joint training of all components of RTDF via backpropagation of the prediction loss. Our empirical evaluation on the benchmark Youtube Face Database (YFDB) [125] and Cambridge-driving Labeled Video Database (CamVid) [9] demonstrates the advantages of performing joint inference and joint training of RTDF, resulting in its superior performance over the state of the art. The results suggest that our end-to-end joint training of all components of RTDF is more critical for accurate video labeling than their joint inference. Also, RTDF performance on a frame can be improved by previously labeling longer sequences of frames preceding that frame. Finally, we have empirically found that RTDF poorly handles abrupt scale changes and labeling of thin, elongated objects.

Chapter 3: Boundary Flow: A Siamese Network that Predicts Boundary Motion without Training on Motion

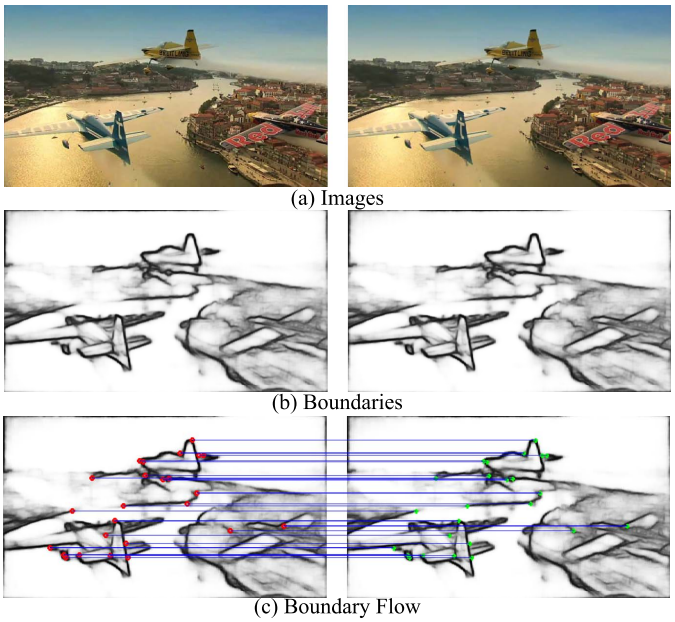


Figure 3.1: Boundary flow estimation. Given two images (a), our approach *jointly*: predicts object boundaries in both images (b), and estimates motion of the boundaries in the two images (c). For clarity, only a part of boundary matches are shown in (c).

3.1 Introduction

This chapter considers the problem of estimating motions of object boundaries in two consecutive video frames, or simply two images. We call this problem boundary flow (BF) estimation. Intuitively, BF is defined as the motion of every pixel along object boundaries in two images, as illustrated in Figure 3.1. A more rigorous definition will be presented in Sec. 3.3. BF estimation is an important problem. Its solution can be used as an informative mid-level visual cue for a wide range of higher-level vision tasks, including

object detection (e.g.,[32]), object proposals (e.g.,[141]), video segmentation (e.g.,[70]), and depth prediction (e.g., [1]). This is because, in a BF, the boundaries identify objects' locations, shapes, motions, local interactions, and figure-ground relationships. In many object-level tasks, BF can be computed in lieu of the regular optical flow, hence avoiding estimating motion on many irrelevant background pixels that may not be essential to the performance of the task.

Yet, this problem has received scant attention in the literature. Related work has mostly focused on single-frame edge detection and dense optical flow estimation. These approaches, however, cannot be readily applied to BF estimation, due to new challenges. In particular, low-level spatiotemporal boundary matching — which is agnostic of objects, scenes, and motions depicted in the two video frames — is subject to many ambiguities. The key challenge is that distinct surfaces sharing a boundary move with different motions, out-of-plane rotations and changing occlusions. This makes appearance along the boundary potentially inconsistent in consecutive frames. The difficulty of matching boundaries in two images also increases when multiple points along the boundary have similar appearance.

Our key hypothesis is that because of the rich visual cues along the boundaries, BF may be learned **without** pixel-level motion annotations, which is typically very hard to come by (prior work resorts to simulations [84] or computer graphics [12], which may not represent realistic images).

While there are a few approaches that separately detect and match boundaries in a video, e.g., [81, 117, 118], to the best of our knowledge, this is the first work that gives a rigorous definition of boundary flow, as well as *jointly* detects object boundaries and estimates their flow within the deep learning framework. We extend ideas from deep boundary detection approaches in images [127, 130], and specify a new Fully Convolutional Siamese encoder-decoder Network (FCSN) for joint spatiotemporal boundary detection and BF estimation. As shown in Figure 3.2, FCSN encodes two consecutive video frames into a coarse joint feature representation (JFR) (marked as a green cube in Figure 3.2). Then, a Siamese decoder uses deconvolution and un-max-pooling to estimate boundaries in each of the two input images.

Our network trains only on boundary annotations in one frame and predicts boundaries in each frame, so at first glance it does not provide motion estimation. However, the Siamese network is capable of predicting different (but correct) boundaries in two

frames, while the only difference in the two decoder branches are max-pooling indices. Thus, our key intuition is that there must be a common edge representation in the JFR layer for each edge, that are mapped to two different boundary predictions by different sets of max-pooling indices. Such a common representation enables us to match the corresponding boundaries in the two images. The matching is done by tracking a boundary from one boundary prediction image back to the JFR, and then from the JFR to boundaries in the other boundary prediction image. This is formalized as an excitation attention-map estimation of the FCSN. We use edgelet-based matching to further improve the smoothness and enforce ordering of pixel-level boundary matching along an edgelet.

Since FCSN performs boundary detection and provides correspondence scores for boundary matching, we say that FCSN *unifies* both boundary detection and BF estimation within the same deep architecture. In our experiments, this approach proves capable of handling large object displacements in the two images, and thus can be used as an important *complementary* input to dense optical flow estimation.

We evaluate FCSN on the VSB100 dataset [33] for boundary detection, and on the Sintel training dataset [12] for BF estimation. Our results demonstrate that FCSN yields higher precision on boundary detection than the state of the art, and using the excitation attention score for boundary matching yields superior BF performance relative to reasonable baselines. Also, experiments performed on the Sintel test dataset show that we can use the BF results to augment the input of a state-of-the-art optical flow algorithm – CPM-Flow [49] – and generate significantly better dense optical flow than the original.

Our key contributions are summarized below:

- We consider the problem of BF estimation within the deep learning framework, give a rigorous definition of BF, and specify and extensively evaluate a new deep architecture FCSN for solving this problem. We also demonstrate the utility of BF for estimating dense optical flow.
- We propose a new approach to generate excitation-based correspondence scores from FCSN for boundary matching, and develop an edgelet-based matching for refining point matches along corresponding boundaries.
- We improve the state-of-the-art on spatiotemporal boundary detection, provide the first results on BF estimation, and achieve competitive improvements on dense

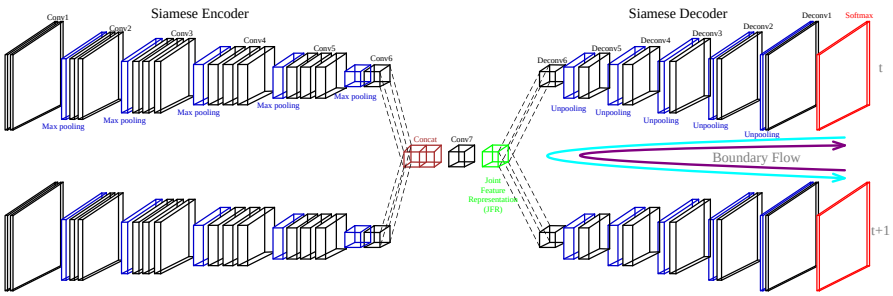


Figure 3.2: FCSN consists of a Siamese encoder and a Siamese decoder and takes two images as input. The two Siamese soft-max outputs of the decoder produce boundary predictions in each of the two input images. Also, the decoder associates the two Siamese branches via the decoder layers and the JFR layer (the green cube) for calculating the excitation attention score, which in turn is used for BF estimation, as indicated by the cyan and purple arrows. The convolution, pooling, softmax and concatenation layers are marked with black, blue, red and brown respectively. Best viewed in color.

optical flow when integrated with CPM-Flow [49].

3.2 Related Work

This section reviews closely related work on boundary detection and dense optical flow estimation. The literature on semantic video segmentation and semantic contour detection is beyond our scope.

Boundary Detection. Traditional approaches to boundary detection typically extract a multitude of hand-designed features at different scales, and pass them to a detector for boundary detection [3]. Some of these methods leverage the structure of local contour maps for fast edge detection [24]. Recent work resort to convolutional neural networks (CNN) for learning deep features that are suitable for boundary detection [35, 102, 8, 7, 127, 130, 83]. [59] trains a boundary detector on a video dataset and achieved improved results. Their network is defined on a single frame and does not provide motion information across two frames. The approach of [130] is closest to ours, since they use a fully convolutional encoder-decoder for boundary detection on one frame. However, without a Siamese network their work cannot be used to estimate boundary motion as proposed in this chapter.

Optical flow estimation. There has been considerable efforts to improve the effi-

ciency and robustness of optical flow estimation, including PatchMatch [6] and extensions [61, 43, 5]. They compute the Nearest Neighbor Field (NNF) by random search and propagation. EpicFlow [95] uses DeepMatching [124] for a hierarchical matching of image patches, and its extension Coarse-to-fine Patch-Match (CPM-Flow) [49] introduces a propagation between levels of the hierarchical matching. While EpicFlow [95] propagates optical flow to image boundaries, it still does not handle very abrupt motions well, as can be seen in many of the fast-moving objects in the Sintel benchmark dataset. In this chapter, we do not focus on dense optical flow estimation, but demonstrate the capability of boundary flow estimation in supplementing optical flow, which is beneficial in large displacements and flow near boundaries. As our results show, we improve CPM-Flow when using our boundary flow estimation as a pre-processing step. Boundary motion estimation was first considered in [81], and then in [129] where dense optical flow was initialized from an optical flow computed on Canny edges. However, in both of these papers, the definition of their edge flow differs from our boundary flow in the following. First, they do not consider cases when optical flow is not defined. Second, they do not have a deep network to perform boundary detection. Finally, they do not evaluate edge flow as a separate problem.

3.3 Boundary Flow

This section defines BF, introduces the FCSN, and specifies finding boundary correspondences in the two frames using the FCSN's excitation attention score.

3.3.1 Definition of Boundary Flow

BF is defined as the motion of every boundary pixel towards the corresponding boundary pixel in the next frame. In the case of out-of-plane rotations and occlusions, BF identifies the occlusion boundary closest to the original boundary pixel (which becomes occluded). We denote the set of boundaries in frame t and $t+1$ as B_1 and B_2, respectively. Let $\mathrm{OF}(\mathbf{x})$ denote the optical flow of a pixel $\mathbf{x}$ in frame t, and $\mathbf{x}+\mathrm{OF}(\mathbf{x})$ represent a mapping of pixel $\mathbf{x}$ in frame $t+1$. Boundary flow $\mathrm{BF}(\mathbf{x})$ is defined as:
(i) $\mathrm{BF}(\mathbf{x}) = \arg\min_{\mathbf{y}\in B_2} \|\mathbf{y} - (\mathbf{x} + \mathrm{OF}(\mathbf{x}))\|_2 - \mathbf{x}$, if $\mathrm{OF}(\mathbf{x})$ exists;
(ii) $\mathrm{BF}(\mathbf{x}) = \mathrm{OF}(\arg\min_{\mathbf{y},\exists \mathrm{OF}(\mathbf{y})} \|\mathbf{y}-\mathbf{x}\|_2)$, if $\mathrm{OF}(\mathbf{x})$ does not exist ($\mathbf{x}$ occluded in frame

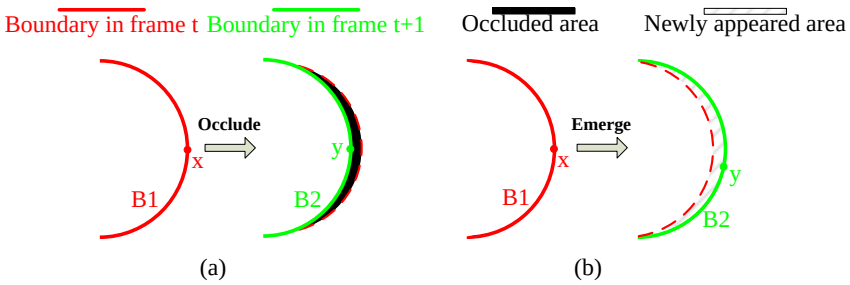


Figure 3.3: Figure 3.3(a) shows the case when a boundary B_1 in frame t is occluded at time $t+1$. Figure 3.3(b) shows the case when a boundary B_1 in frame t is no longer a boundary at time $t+1$ but its pixels are visible. In both cases BF is well-defined and always resides on the boundary.

$t+1$);
(*iii*) BF($\mathbf{x}$) is undefined if *argmin* in (*i*) or (*ii*) does not return a unique solution.

In (*i*), BF is defined as optical flow for translations and elastic deformations, or the closest boundary pixel from the optical flow for out-of-plane rotations (see Figure 3.3(b)). In (*ii*), BF is defined as the closest occlusion boundary of the pixel which becomes occluded (see Figure 3.3(a)). Thus, BF can be defined even if optical flow is not defined. Since optical flow is often undefined in the vicinity of occlusion boundaries, BF captures shapes/occlusions better than optical flow. In (*iii*), BF is undefined only in rare cases of fast movements with symmetric occluders (e.g. a perfect ball) resulting in multiple pixels as the argmin solution.

3.3.2 Fully Convolutional Siamese Network

We formulate boundary detection as a binary labeling problem. For this problem, we develop a new, end-to-end trainable FCSN, shown in Figure 3.2. FCSN takes two images as input, and produces binary soft-max outputs of boundary predictions in each of the two input images. The fully convolutional architecture in FCSN scales up to arbitrary image sizes.

FCSN consists of two modules: a Siamese encoder, and a Siamese decoder. The encoder stores all the pooling indices and encodes the two frames as the joint feature representation (JFR) (green box in Figure 3.2) through a series of convolution, ReLU,

and pooling layers. The outputs of the encoder are concatenated, and then used as the input to the decoder. The decoder takes both the JFR and the max-pooling indices from the encoder as inputs. Then, the features from the decoder are passed into a softmax layer to get the boundary labels of all pixels in the two images.

The two branches of the encoder and the two branches of the decoder use the same architecture and share weights with each other. However, for two different input images, the two branches would still output different predictions, since decoder predictions are modulated with different pooling indices recorded in their corresponding encoder branches. Each encoder branch uses the layers of VGG net [105] until the *fc6* layer. The decoder decodes the JFR to the original input size through a set of unpooling, deconvolution, ReLU and dropout operations. Unlike the deconvolutional net [88] which uses a symmetric decoder as the encoder, we design a light-weight decoder with fewer weight parameters than a symmetric structure for efficiency. Except for the layer right before the *softmax* layer, all the other convolution layers of the decoder are followed by a ReLU operator and a dropout layer. A detailed description of the convolution and dropout layers is summarized in Table 3.1.

Layer	Filter	Dropout rate
Deconv1	$1 \times 1 \times 512$	0.5
Deconv2	$5 \times 5 \times 512$	0.5
Deconv3	$5 \times 5 \times 256$	0.5
Deconv4	$5 \times 5 \times 128$	0.5
Deconv5	$5 \times 5 \times 64$	0.5
Deconv6	$5 \times 5 \times 32$	0.5
Softmax	$5 \times 5 \times 1$	-

Table 3.1: The configuration of the decoder in FCSN.

3.3.3 Boundary Flow Estimation

This section first describes estimation of the excitation attention score, used as a cue for boundary matching, and then specifies our edgelet-based matching for refining point matches along the boundaries.

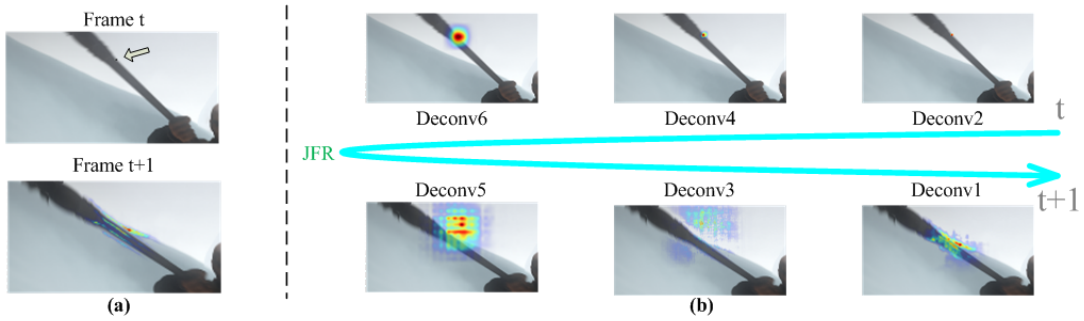


Figure 3.4: (a) Estimation of the excitation attention score in frame $t+1$ (bottom) for a particular boundary point in frame t (top; the point is indicated by the arrow). The attention map is well-aligned with the corresponding boundary in frame $t+1$, despite significant motion. (b) Visualization of attention maps at different layers of the decoders of FCSN along the excitation path (cyan) from a particular boundary point in frame t to frame $t+1$ via the JFR. For simplicity, we only show the attention maps in some of the layers from the decoder branch at time t and $t+1$. As can be seen, starting from a pixel on the predicted boundary in frame t, the attention map gradually becomes coarser along the path to the JFR. Then from the JFR to boundary prediction in frame $t+1$, the excitation attention scores gradually become refined and more focused on the most relevant pixels in frame $t+1$. (Best viewed in color)

3.3.3.1 Excitation Attention Score

A central problem in BF estimation is to identify the correspondence between a pair of boundary points $\langle \mathbf{x}_t^i, \mathbf{y}_{t+1}^j \rangle$, where $\mathbf{x}_t^i$ is a boundary point in frame t, and $\mathbf{y}_{t+1}^j$ is a boundary point in frame $t+1$. Our key idea is to estimate this correspondence by computing the excitation attention scores in frame $t+1$ for every $\mathbf{x}_t^i$ in frame t, as well as the excitation attention scores in frame t for every $\mathbf{y}_{t+1}^j$ in frame $t+1$. The excitation attention scores can be generated efficiently using excitation backpropagation (ExcitationBP) [137] – a probabilistic winner-take-all approach that models dependencies of neural activations through convolutional layers of a neural network for identifying relevant neurons for prediction, i.e., attention maps.

The intuition behind our approach is that the JFR stores a joint representation of two corresponding boundaries of the two images, and thus could be used as a "bridge" for matching them. This "bridge" is established by tracking the most relevant neurons

along the path from one branch of the decoder to the other branch via the JFR layer (the cyan and purple arrows in Figure 3.2).

In our approach, the winner neurons are sequentially sampled for each layer on the path from frame t to $t+1$ via the JFR, based on a conditional winning probability. The relevance of each neuron is defined as its probability of being selected as a winner on the path. Following [137], we define the winning probability of a neuron a_m as

$$\begin{aligned} p(a_m) &= \sum_{n \in \mathcal{P}_m} p(a_m|a_n)p(a_n) \\ &= \sum_{n \in \mathcal{P}_m} \frac{w^+_{mn} a_m}{\sum_{m' \in \mathcal{C}_n} w^+_{m'n} a_{m'}} p(a_n) \end{aligned} \tag{3.1}$$

where $w^+_{mn} = max\{0, w_{mn}\}$, $\mathcal{P}_m$ and $\mathcal{C}_n$ denote the parent nodes of a_m and the set of children of a_n in the path traveling order, respectively. For our path that goes from the prediction back to the JFR layer, $\mathcal{P}_m$ refers to all neurons in the layer closer to the prediction, and $\mathcal{C}_n$ refers to all neurons in the layer closer to the JFR layer.

ExcitationBP efficiently identifies which neurons are responsible for the final prediction. In our approach, ExcitationBP can be run in parallel for each edgelet (see next subsection) of a predicted boundary. Starting from boundary predictions in frame t, we compute the marginal winning probability of all neurons along the path to the JFR. Once the JFR is reached, these probabilities are forward-propagated in the decoder branch of FCSN for finally estimating the pixel-wise excitation attention scores in frame $t+1$. For a pair of boundary points, we obtain the attention score $s_{i \to j}$. Conversely, starting from boundary predictions in frame $t+1$, we compute the marginal winning probability of all neurons along the path to JFR, and feed them forward through the decoder for computing the excitation attention map in frame t. Then we can obtain the attention score $s_{j \to i}$. The attention score between a pair of boundary points $\langle \mathbf{x}^i_t, \mathbf{y}^j_{t+1} \rangle$ is defined as the average of $s_{i \to j}$ and $s_{j \to i}$, which we denote as s_{ij}. An example of our ExcitationBP is shown in Figure 3.4.

3.3.3.2 Edgelet-based Matching

After estimating the excitation attention scores s_{ij} of boundary point pairs $\langle \mathbf{x}^i_t, \mathbf{y}^j_{t+1} \rangle$, as described in Sec. 3.3.3.1, we use them for matching corresponding boundaries that have

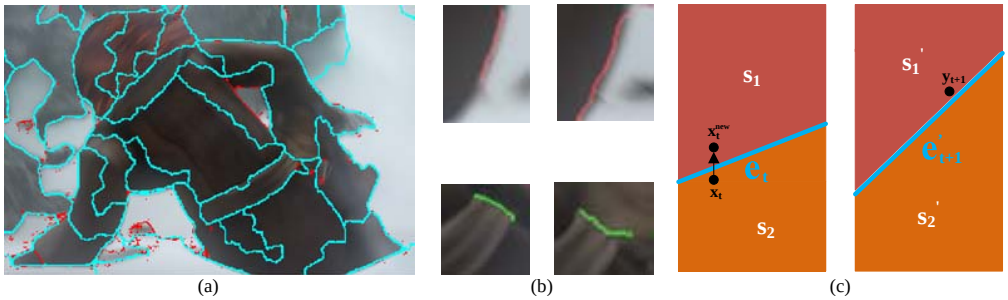


Figure 3.5: Overview of edgelet matching. The matching process consists of three phases: superpixel generation, edgelet matching, and flow placement. The two frames are first over-segmented into large superpixels using the FCSN boundaries. (a) most of the boundary points (in red color) are well aligned with the superpixel boundaries (in cyan color); (b) Example edgelet matches. In the second case, it can be seen clearly that the appearance only matches on one side of the edgelet. (c) The process of matching and flow placement. Sometimes, because of the volatility of edge detection, $\mathbf{x}_t$ and $\mathbf{y}_{t+1}$ falls on different sides of the boundary, we will need to then move $\mathbf{x}_t$ so that they fall on the same side. Note that s_1 and s_2, $s_1^{'}$ and $s_2^{'}$ denote the superpixel pairs falling on the two sides of the edgelets.

been predicted in frames t and $t+1$. While there are many boundary matching methods that would be suitable, in this work we use the edgelet-based matching which not only finds good boundary correspondences, but also produces the detailed point matches along the boundaries, as needed for our BF estimation. To this end, we first decompose the predicted boundaries into smaller edgelets, then apply edgelet-based matching to pairs of edgelets.

From predicted boundaries to edgelets. Given the two input images and their boundary predictions from FCSN, we oversegment the two frames using sticky superpixels [24], and merge the superpixels to larger regions as in [51]. Importantly, both oversegmentation and superpixel-merging use our boundary predictions as input, ensuring that contours of the resulting regions strictly respect our predicted boundaries, as illustrated in Figure 3.5(a). We define an edgelet as all the points that lie on a given boundary shared by a pair of superpixels. Figure 3.5(b) shows two examples of matching edgelet pairs in frames t and $t+1$.

Edgelet matching. We apply edgelet-based matching to each edgelet pair, e_t in frame

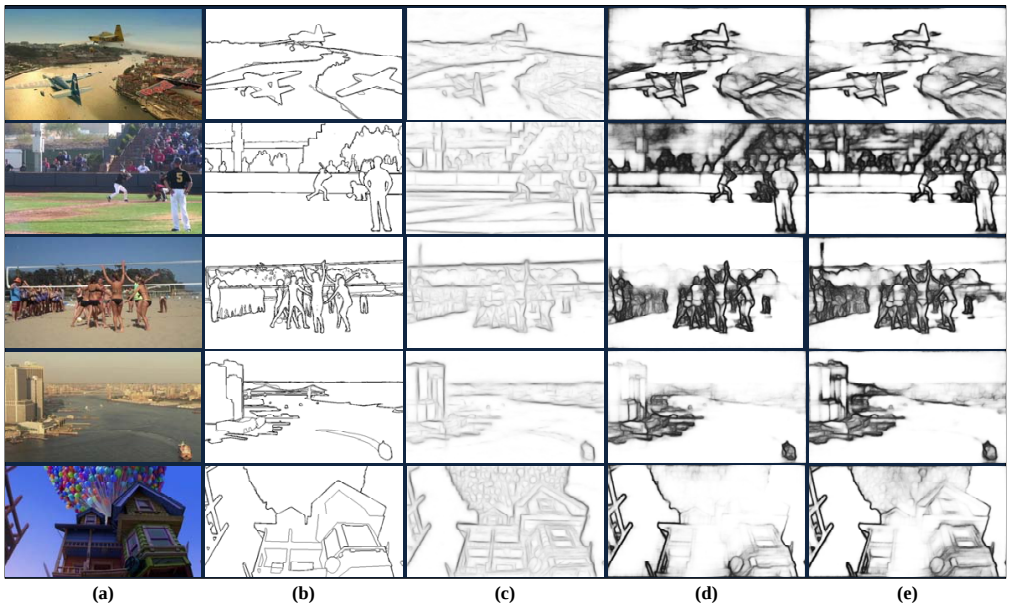


Figure 3.6: Example results on VSB100. In each row from left to right we present (a) input image, (b) ground truth annotation, (c) edge detection [24], (d) object contour detection [130] and (e) our boundary detection.

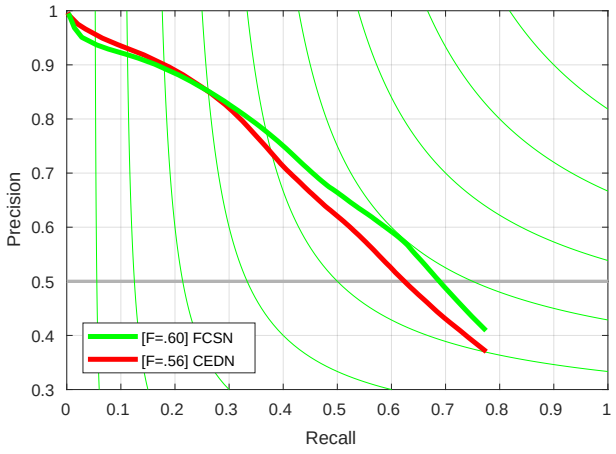


Figure 3.7: PR curve for object boundary detection on VSB100.

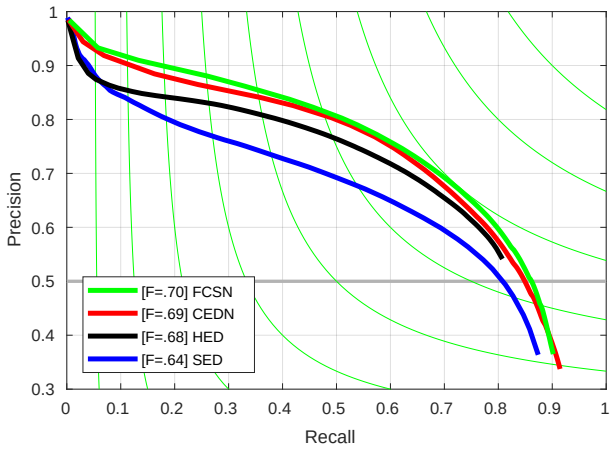


Figure 3.8: PR curve for object boundary detection on VSB100 with fine-tuning on both BSDS500 and VSB100 training sets.

t and e'_{t+1} in frame $t+1$, that fall within a reasonable spatial neighborhood (empirically set to 100 pixels around the edgelet as sufficient to accommodate for large motions). For each edgelet pair, e_t in frame t and e'_{t+1} in frame $t+1$, all the similarities between all the pixel pairs on these edgelet pairs are summed up and normalized to obtain the similarity between the edgelet pair. The similarity between points $\langle \mathbf{x}_t^i, \mathbf{y}_{t+1}^j \rangle$ on e_t and e'_{t+1} is expressed in terms of their respective excitation attention scores as s_{ij}.

For an edgelet e_t in frame t, we keep the top-10 most similar edgelets in frame $t+1$ as its matching candidates. These candidate edgelet pairs are further filtered by their normals, with only edgelets with an angle not more than 45 degrees retained. The normals are computed as the average direction from pixel coordinates on one side of the edge to corresponding pixel coordinates on the other side of the edge. This also helps to determine which superpixel pair falls on the same side of the edge in the two images. As shown in Figure 3.5(c), superpixels s_1 and s'_1 fall on the left side of edges e_t and e'_{t+1}, respectively, thus superpixel pair $\{s_1, s'_1\}$ fall on the same side of the edges.

After filtering by angle, a greedy matching algorithm is performed to approximate bipartite matching of edgelets in frame t to edgelets in the frame $t+1$. This further reduces the number of edgelet pairs retained.

For the final boundary flow placement, we observe that some boundary points will

Method	ODS	OIS	AP
CEDN [130]	0.563	0.614	0.547
FCSN	**0.597**	**0.632**	**0.566**

Table 3.2: Results on VSB100.

be placing on the incorrect side of the edgelet. We utilize normalized region similarity defined by color to assign the motion to superpixels pairs that are more similar to each other in color. As shown in Figure 3.5(c), point $\mathbf{x}_t$ is on the right side of edge e_t but the corresponding point $\mathbf{y}_{t+1}$ is on the left side of edge e'_{t+1}. Our approach moves $\mathbf{x}_t$ to the other side of e_t, resulting in $\mathbf{x}_t^{new}$. After moving the points, we obtain pixel-level matches which are the final boundary flow result.

3.4 Training

FCSN is implemented using Caffe [54]. The encoder weights are initialized with VGG-16 net and fixed during training. We update only the decoder parameters using the Adam method [60] with learning rate 10^{-4}. We train on VSB100, a state-of-the-art video object boundary dataset, which contains 40 training videos with annotations on every 20-th frame, for a total of 240 annotated frames. Because there are too few annotations, we augment the training with the PASCAL VOC 12 dataset, which contains 10582 still images (with refined object-level annotations as in [130]). In each iteration, 8 patches with size 224×224 are randomly sampled from an image pair of VSB100 (or two duplicated frames of PASCAL VOC) and passed to the model.

The loss function is specified as the weighted binary cross-entropy loss common in boundary detection [127] $L(w) = -\frac{1}{N}\sum_{i=1}^{N}[\lambda_1 y_n \log \hat{y}_n + \lambda_2(1-y_n)\log(1-\hat{y}_n)]$ where N is the number of pixels in an iteration. Note that the loss is defined on a single side of the outputs, since only single frame annotations are available. The two decoder branches share the same architecture and weights, and thus can be both updated simultaneously with our one-side loss. The two branches still can output different predictions, since decoder predictions are modulated with different pooling indices recorded in the corresponding encoder branches. Due to the imbalances of boundary pixels and non-boundary pixels, we set λ_1 to 1 and λ_2 to 0.1, respectively.

Method	ODS	OIS	AP
SE [24]	0.643	0.680	0.608
HED [127]	0.677	0.715	0.618
CEDN [130]	0.686	0.718	0.687
FCSN	**0.698**	**0.729**	**0.705**

Table 3.3: Results on VSB100 with fine-tuning on both BSDS500 and VSB100 training sets.

Method	FLANN [87]	RANSAC [10]	Greedy	Our Matching
EPE	23.158	20.874	25.476	9.856

Table 3.4: Quantitative results of boundary flow on Sintel training dataset in EPE metric.

3.5 Results

This section presents our evaluation of boundary detection, BF estimation, and utility of BF for optical flow estimation.

3.5.1 Boundary Detection

After FCSN generates boundary predictions, we apply the standard non-maximum suppression (NMS). The resulting boundary detection is evaluated using precision-recall (PR) curves and F-measure.

VSB100. For the benchmark VSB100 test dataset [33], we compare with the state-of-the-art approach CEDN [130]. We train both FCSN and CEDN using the same training data with 30000 iterations. Note that CEDN is single-frame based. Nevertheless, both FCSN and CEDN use the same level of supervision, since only isolated single frame annotations apart from one another are available. Figure 3.7 shows the PR-curves of object boundary detection. As can be seen, F-score of FCSN is 0.60 while 0.56 for CEDN. FCSN yields higher precision than CEDN, and qualitatively we observe that FCSN generates visually cleaner object boundaries. As shown in Figure 3.6, CEDN misses some of the boundaries of background objects, but our FCSN is able to detect them. Due to limited training data, both FCSN and CEDN obtain relatively low recall. Table 3.2 shows that FCSN outperforms CEDN in terms of the optimal dataset scale (ODS), optimal image scale (OIS), and average precision (AP).

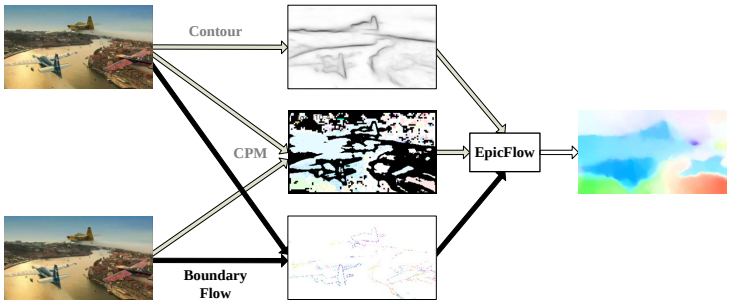


Figure 3.9: Overview of augmenting boundary flow into the framework of CPM-Flow. Given two images, we compute the standard input to CPM-Flow: matches using CPM matching [49] and the edges of the first image using SE [24]. Then we augment the matches with our predicted boundary flow (i.e., matches on the boundaries), as indicated by black arrows.

Finetuning on BSDS500 and VSB100. We also evaluate another training setting when FCSN and CEDN are both fine-tuned on the BSDS500 training dataset [3] and VSB100 training set for 100 epochs with learning rate 10^{-5}. BSDS has more edges annotated hence allows for higher recall. Such trained FCSN and CEDN are then compared with the state-of-the-art, including structured edge detection (SE) [24], and holistically-nested edge detection algorithm (HED) [127]. Both SE and HED are re-trained with the same training setting as ours. Figure 3.8 and Table 3.3 present the PR-curves and AP. As can be seen, FCSN outperforms CEDN, SE and HED in all metrics. We presume that further improvement may be obtained by training with annotated boundaries in both frames.

3.5.2 Boundary Flow Estimation

Boundary flow accuracies are evaluated by average end-point error (EPE) between our boundary flow prediction and the ground truth boundary flow (as defined in Sec. 3.3.1) on the Sintel training dataset.

In order to identify a good competing approach, we have tested a number of the state-of-art matching algorithms on the Sintel training dataset, including coarse-to-fine PatchMatch (CPM) [49], Kd-tree PatchMatch [43] and DeepMatching [124], but have

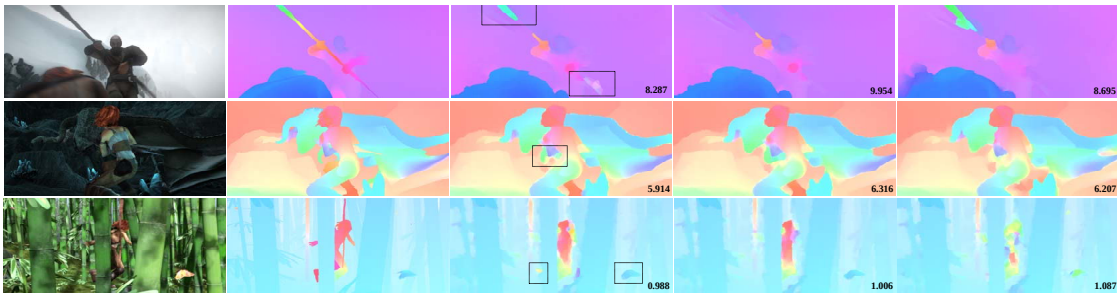


Figure 3.10: Example results on MPI-Sintel test dataset. The columns correspond to original images, ground truth, CPM-AUG (i.e., our approach), CPM-Flow [49] and EpicFlow[95]. The rectangles highlight the improvements and the numbers indicate the EPEs.

found that these algorithms are not suitable for our comparison because they prefer to find point matches off boundaries.

Therefore, we compare our edgelet-based matching algorithm with the following baselines: (i) greedy nearest-neighbor point-to-point matching, (ii) RANSAC [10], (iii) FLANN, a matching method that uses SIFT feautres. The quantitative results are summarized in Table 3.4. Our edgelet-based matching outperforms all the baselines significantly.

3.5.3 Dense Optical Flow Estimation

We also test the utility of our approach for optical flow estimation on the Sintel testing dataset. After running our boundary flow estimation, the resulting boundary matches are used to augment the standard input to the state of the art CPM-Flow [49], as shown in Figure 3.9. Such an approach is denoted as CPM-AUG, and compared with the other existing methods in Table 3.5. As can be seen, CPM-AUG outperforms CPM-Flow and FlowFields. Note the results we submitted on "Sintel clean" under the name CPM-AUG was not actually results of CPM-AUG, actually it was just our implementation of CPMFlow[49], which is a bit lower than the public one on the Sintel dataset. However these are the best results we can obtain using the public implementation of the algorithm. In principle, the augmented point matches should be able to help other optical flow

Method	EPE all	EPE matched	EPE unmatched
CPM-AUG	**5.645**	2.737	**29.362**
FlowFields[5]	5.810	**2.621**	31.799
Full Flow[14]	5.895	2.838	30.793
CPM-Flow[49]	5.960	2.990	30.177
DiscreteFlow[85]	6.077	2.937	31.685
EpicFlow[95]	6.285	3.060	32.564

Table 3.5: Quantitative results on Sintel final test set.

algorithms as well as it is largely orthogonal to the information pursued by current optical flow algorithms.

Figure 3.10 shows qualitative results of CPM-AUG on Sintel testing dataset with comparison to two state-of-the-art methods: CPM-Flow and EpicFlow. As it can be seen, CPM-AUG performs especially well on the occluded areas and benefits from the boundary flow to produce sharp motion boundaries on small objects like the leg and the claws as well as the elongated halberd.

3.6 Summary

We have formulated the problem of boundary flow estimation in videos. For this problem, we have specified a new end-to-end trainable FCSN which takes two images as input and produces boundary detections in each image. We have also used FCSN to generate excitation attention maps in the two images as informative features for boundary matching, thereby unifying detection and flow estimation. For matching points along boundaries, we have decomposed the predicted boundaries into edgelets and applied edgelet-based matching to pairs of edgelets from the two images. Our experiments on the benchmark VSB100 dataset for boundary detection demonstrate that FCSN is superior to the state- of-the-art, succeeding in detecting boundaries both of foreground and background objects. We have presented the first results of boundary flow on the benchmark Sintel training set, and compared with reasonable baselines. The utility of boundary flow is further demonstrated by integrating our approach with the CPM-Flow for dense optical flow estimation. This has resulted in an improved performance over the

original CPM-Flow, especially on small details, sharp motion boundaries, and elongated thin objects in the optical flow.

Chapter 4: Temporal Deformable Residual Networks for Action Segmentation in Videos

4.1 Introduction

In this chapter, we address action segmentation where the goal is to label video frames with appropriate action classes. Action segmentation is a basic vision problem, and of great importance to a wide range of applications, including video surveillance and robot navigation.

Recent approaches typically address this problem in two steps: i) Extraction of spatial or spatiotemporal features using convolutional neural networks, e.g., two-stream CNNs [104] or local 3D ConvNets [120], and ii) Classification of the extracted features using a one-directional model, e.g., encoder-decoder temporal convolutional networks (ED-TCN) [65], or bi-directional LSTM networks (Bi-LSTM) [106, 50]. Although recurrent deep models have shown promise in capturing latent temporal patterns [106, 50, 23], they are hard to train [90], and have a limited span of attention [106].

Toward overcoming these limitations, we present a new temporal convolutional model, named temporal deformable residual network (TDRN). TDRN classifies every video frame using a deep temporal residual network. The residual network takes frame-level CNN features as input and computes deformable convolutions along time at multiple temporal scales, starting at the frame-level resolution. As shown in Figure 4.1, this computation is done in two parallel temporal streams: i) Residual stream that analyzes video information at its full temporal resolution, and ii) Pooling/unpooling stream that captures long-range video information at different scales. The first stream is aimed at resolving ambiguities about local, frame-to-frame segmentation, and the second stream uses multiscale context for improving accuracy of frame classification. The temporal residual stream and the temporal pooling stream are fused through a set of deformable temporal residual modules (DTRMs), and coupled with temporal residuals at the full temporal resolution. In addition, TDRN computes deformable temporal convolutions for modeling variations in temporal extents of human actions, similar to deformable spatial

convolution that has been shown to improve object detection in images [20].

As our results demonstrate, the two-stream residual computation and deformable temporal convolutions make TDRN more robust against temporal transformations than recent deep networks, including encoder-decoder temporal convolutional networks (ED-TCNs) [68, 65], temporal convolutional U-networks (TUNets) [98], and temporal residual networks (TResNets) [44], illustrated in Figure 4.2. As can be seen in Figure 4.2, ED-TCNs use a sequence of regular temporal convolutions and temporal pooling/unpooling layers within a single processing stream. TUNets simply concatenate features computed in their unpooling path with the corresponding features at the same temporal scale from the pooling path, as indicated by the cyan arrows. TResNets add shortcut connections (marked brown) between a layer and its succeeding layer for allowing the gradients to propagate more effectively through the network in learning. None of these models use deformable temporal convolutions and two processing streams since they all compute regular temporal convolutions in a single processing stream. Unlike these related models, we use two processing streams and deformable temporal convolutions, enabling robust action classification and accurate action segmentation in videos.

We evaluate TDRN on the following benchmark datasets: University of Dundee 50 Salads (50Salads), Georgia Tech Egocentric Activities (GTEA) and JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS). Our results demonstrate that TDRN is capable of accurately capturing action durations and transitions between distinct actions. Also, TDRN outperforms the state of the art in frame-wise segmentation accuracy, segmental edit score, and segmental overlap F1 score.

Our key contributions include:

- A new fully-convolutional temporal residual network that consists of two processing streams aimed at extracting both multiscale temporal abstractions and frame-level features for reliable action recognition and precise action segmentation.
- We are not aware of any prior work that uses deformable temporal convolutions; we show they improve action segmentation over regular temporal convolutions.
- We outperform the state of the art in action segmentation on the 50Salads, GTEA and JIGSAWS datasets.

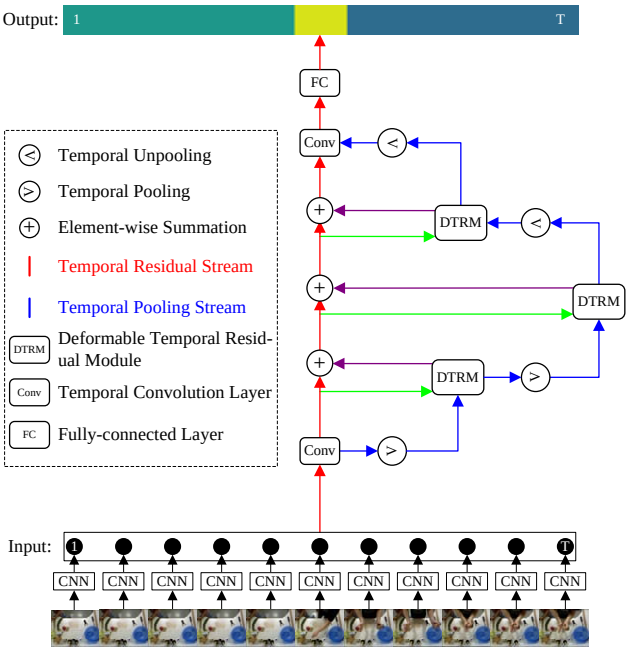


Figure 4.1: For action segmentation, TDRN takes frame-level CNN features as input and outputs frame-wise action labels. TDRN computes two processing streams: Residual stream (marked red) that analyzes video information at its full temporal resolution for precise action segmentation, and Pooling/unpooling stream (marked blue) that captures temporal context at different scales for accurate action recognition. The two streams are fused through a set of deformable temporal residual modules (DTRMs). Best viewed in color.

4.2 Related Work

This section reviews the most related work for action segmentation and detection in which most of them are about temporal modeling. A host of work on spatiotemporal modeling for video recognition [113, 58, 122, 120, 104, 135, 19, 30, 31, 38, 123] and action detection [103, 57, 107, 48, 22] are beyond the scope of this chapter.

Action Segmentation. Existing approaches typically first extract frame-level features, and then pass them to a temporal model for frame labeling. For example, Yeung et al. [131] use an attention LSTM network to model feature dependencies over a fixed temporal interval. Singh et al. [106] present a multi-stream bi-directional recurrent neu-

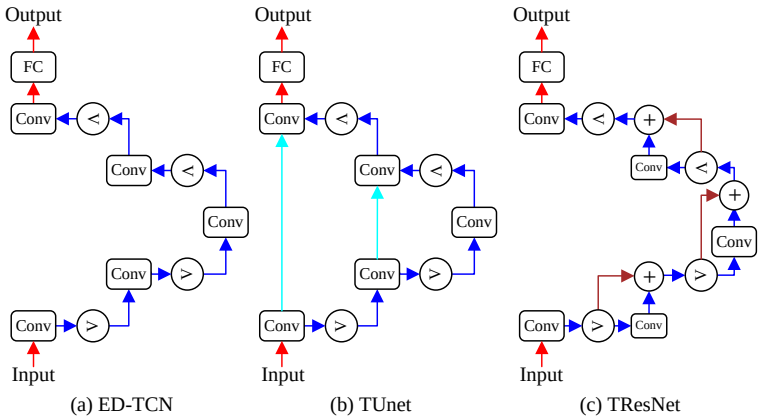


Figure 4.2: Deep architectures of recent work: (a) Encoder-decoder temporal convolutional networks (ED-TCNs) [68], (b) Temporal convolutional U-networks (TUNets) [98], (c) Temporal residual networks (TResNets) [44]. A comparison of these architectures with our TDRN shown in Figure 4.1 makes our differences obvious: none of these models use deformable temporal convolutions and two processing streams. Best viewed in color.

ral network for fine-grained action detection. Fathi et al. [27, 29, 28] use a segmental model that captures object states at the action's start and end. Richard et al. [96] resort to a statistical language model for representing temporal and contextual structure in videos of varying lengths. Kuehne et al. [62] use Hidden Markov Models (HMMs) on dense-trajectory features and propose an end-to-end generative approach for action segmentation.

The approach of Lea et al. [65] is the most related to ours, as they use two temporal convolutional networks for action segmentation and detection. However, their model computes regular temporal convolutions in a single processing stream, whereas our TDRN computes deformable temporal convolutions in two temporal streams. Ding et al. [23] replace the convolutional decoder in the approach of Lea et al. [65] with a bi-directional LSTM (Bi-LSTM) [39]. However, their network is a hybrid of temporal convolutional network and temporal recurrent network, and thus inherits the well-known limitations of recurrent models, including difficult training [90], and limited attention span [106].

Action Detection. A number of approaches to action detection is also related to

ours. For example, for action detection, (a) Multi-region two-stream network [91] links frame-level detections of the faster R-CNN [94]; (b) Recurrent models are learned from 3D skeleton data [77] and under weak supervision [97]; (c) Reinforcement learning is used for predicting temporal bounds of actions based on observing only a fraction of the video [132]; (d) Structured temporal pyramid models the temporal structure of actions [139]; (e) Region convolutional 3D network (R-C3D) with 3D ROI pooling encodes video streams [128]; (f) Flow network searches for temporal intervals with a maximum sum of frame-wise classification scores [134]; (g) Temporal convolutional model extracts context of action proposals through a pair-wise sampling layer [21]; and (h) Temporal single shot action detector network detects action instances [79].

In some of the aforementioned approaches, video features are usually sampled at two temporal scales for generating good action proposals. Also, some of the approaches consist of modules that are typically not jointly trained end-to-end. In contrast, our TDRN fuses multiscale temporal abstractions with features extracted at the frame-wise temporal scale, and can be trained in an end-to-end fashion.

There are some similarities between TDRN and recent work on semantic image segmentation [93], which uses a two-stream spatial residual network to compute pixel-level semantic labeling in the image. In contrast, TDRN computes temporal residual convolutions, which are additionally deformable [20], i.e., capable of modeling variations of temporal extents of actions via deformable temporal convolutions. Hence, we extend [93, 20] from the spatial to temporal domain, where TDRN also analyzes multiple temporal scales.

4.3 Temporal Deformable Residual Network

TDRN computes the residual and pooling/unpooling streams in parallel. As shown in Figure 4.1, features along the residual stream are computed using a sequence of residuals at the full temporal resolution, whereas features in the temporal pooling stream are computed at coarser temporal scales by a sequence of deformable temporal convolutions followed by temporal pooling and corresponding unpooling. The residual stream operates at the finest temporal scale for accurately localizing action boundaries. The temporal pooling/unpooling stream computes contextual features at multiple temporal scales for accurate action recognition using a sequence of deformable temporal residual modules

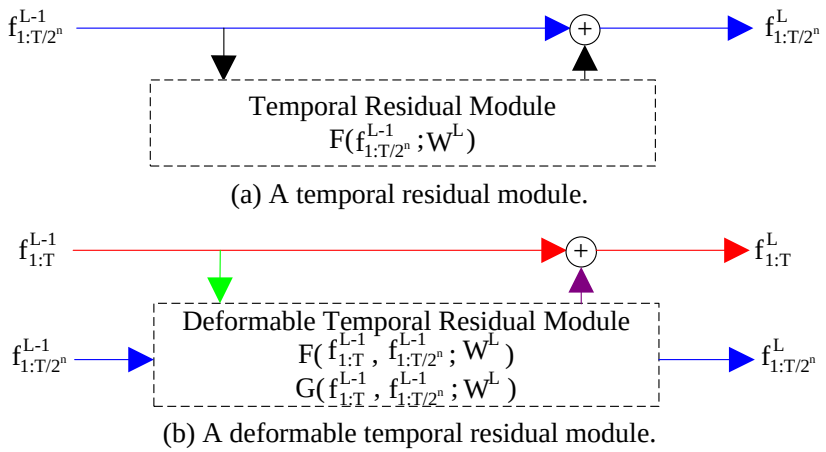


Figure 4.3: Key differences between: (a) Common temporal residual module with a single input and output; and (b) Our deformable temporal residual module (DTRM) with two inputs and two outputs. The input represents feature sequences $f^{L-1}_{1:T/2^n}$ and $f^{L-1}_{1:T}$ with temporal lengths of $T/2^n$ and T, which are computed by the previous layer $L-1$. In (a), the output features are computed by a standard temporal convolution, $F(f^{L-1}_{1:T/2^n}; W^L)$. In (b), the output is computed using a cascade of a deformable temporal convolution, $G(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L)$, followed by a convolution and unpooling, $F(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L)$.

(DTRMs).

Figure 4.3 shows key differences between a common temporal residual module and our DTRM. The former has only one input and one output, while DTRM has two inputs and two outputs. This is because DTRM simultaneously operates on both the residual and pooling streams.

DTRM takes as input two feature sequences, $f^{L-1}_{1:T/2^n}$ and $f^{L-1}_{1:T}$, with temporal lengths of $T/2^n$ and T, which are computed by the previous layer $L-1$ in TDRN. Specifically, $f^{L-1}_{1:T/2^n}$ is produced by the pooling stream and $f^{L-1}_{1:T}$ comes from the residual stream of $L-1$ layer. Note that the layer number L is correlated with the temporal scale n at which the features are computed in TDRN. These are depicted in Figure 4.1 as "vertical" and "horizontal" processing levels in TDRN, respectively. For computing the output feature sequences, $f^{L}_{1:T}$ and $f^{L}_{1:T/2^n}$, DTRM uses a deformable temporal convolution, $G(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L)$, followed by a convolution and unpooling, $F(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L)$,

as

$$\begin{aligned} f^{L}_{1:T} &= f^{L-1}_{1:T} + F(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L) \\ f^{L}_{1:T/2^n} &= G(f^{L-1}_{1:T}, f^{L-1}_{1:T/2^n}; W^L) \end{aligned} \tag{4.1}$$

where W^L denotes network parameters. In (4.1), the output of G is input to F to produce the residual stream.

It is worth noting that our TDRN has similar training characteristics as ResNet [44], since losses can be easily propagated back to the input through the residual stream. In the following section, we describe DTRM in greater detail.

4.3.1 Deformable Temporal Residual Module

The pooling and residual streams in our TDRN are fused through a sequence of DTRMs. Figure 4.4 illustrates the architecture of DTRM. DTRM takes as input the residual and pooling features of the previous layer, $f^{L-1}_{1:T}$ and $f^{L-1}_{1:T/2^n}$, where the pooling sequence of features is computed at a coarser temporal resolution, n, and hence is 2^n times shorter in time than the residual sequence of features. DTRM first applies temporal pooling to $f^{L-1}_{1:T}$, and then concatenates the result with $f^{L-1}_{1:T/2^n}$. The concatenated features are then processed by the deformable temporal convolution module, explained in greater detail in Sec. 4.3.2, resulting in the output pooling features, $f^{L}_{1:T/2^n}$, of the same length as the corresponding input pooling features $f^{L-1}_{1:T/2^n}$. From $f^{L}_{1:T/2^n}$, DTRM computes output residual features, $f^{L}_{1:T}$, using the 1×1 temporal convolution and temporal unpooling.

4.3.2 Deformable Temporal Convolution Module

The deformable temporal convolution module is aimed at improving standard fixed-structure temporal convolutions in modeling temporal variations of action boundaries along the video. It consists of a deformable temporal convolution layer followed by a Normalized Rectified Linear Unit (NRLU) [65], defined as follows:

$$NRLU(\cdot) = \frac{ReLU(\cdot)}{max(ReLU(\cdot)) + \epsilon} \tag{4.2}$$

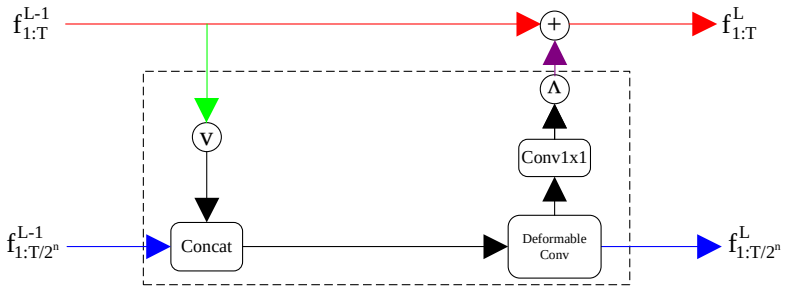


Figure 4.4: DTRM: Given the residual and pooling features of the previous layer as inputs, DTRM applies pooling to the residual stream and then concatenates the result with the input pooling stream. The concatenated features are then processed by the deformable temporal convolution, resulting in the output pooling features, $f^L_{1:T/2^n}$. Also, a temporal residual is computed from $f^L_{1:T/2^n}$ by the 1×1 temporal convolution and temporal unpooling, resulting in output residual features, $f^L_{1:T}$.

where $ReLU$ represents a ReLU layer, $max(\cdot)$ returns the maximal ReLU activation within the layer, and $\epsilon = 0.00001$. Below, we specify deformable temporal convolution.

A temporal convolution can be decomposed into two steps: sampling of input features at specified moments in time, and weighted summation of the sampled features. Analogous to deformable spatial convolutions of objects in images [20], in our approach, the temporal sampling locations that specified by the convolutional kernel are augmented with variable temporal offsets, which in turn are learned end-to-end along with the other network parameters.

Let $\mathcal{I}$ denote the time interval of input feature map f_{in}, and W denote convolution weights. Note that $\mathcal{I}$ defines the temporal receptive field size as well as the dilation size. The temporal convolution consists of sampling over $\mathcal{I}$ and summing the weighted sampled values with weights W as

$$f_{\text{out}}(t_0) = \sum_{t_i \in \mathcal{I}} W(t_i) \cdot f_{in}(t_0 + t_i). \tag{4.3}$$

Deformable convolution specifies a set of offsets $\triangle = \{\triangle t_i | i = 1, 2, \cdots, |\mathcal{I}|\}$, and

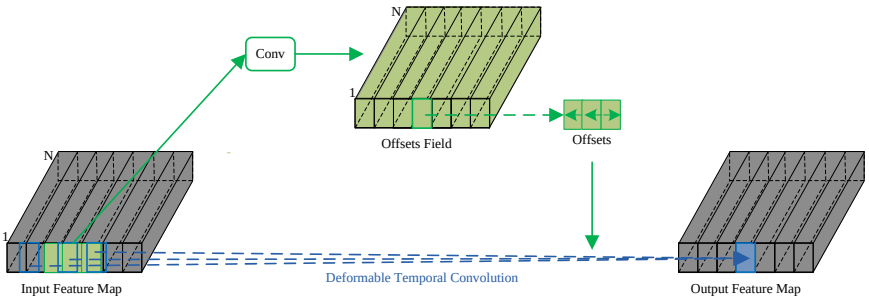


Figure 4.5: An illustration of deformable temporal convolution with kernel size 3 and dilation size 1. Both the temporal offsets and output features are obtained by applying a temporal convolutional layer over the same input feature maps. The offset fields have the same size as the input feature map.

augments the temporal sampling in (4.3) as

$$f_{\text{out}}^{\text{deform}}(t_0) = \sum_{t_i \in \mathcal{I}} W(t_i) \cdot f_{in}(t_0 + t_i + \triangle t_i). \tag{4.4}$$

From (4.4), the sampling is defined over variable time moments $t_i + \triangle t_i$.

Note that $\triangle t_i$ is typically learned as a real number. Thus, $t_0 + t_i + \triangle t_i$ is also a real number. We identify the set of nearest integer temporal locations to $t_0 + t_i + \triangle t_i$ in the feature map, and use bilinear temporal interpolation to compute the ith input feature for the summation in (4.4).

As illustrated in Figure 4.5, the temporal offsets $\{\triangle t_i\}$ are obtained by applying a temporal convolutional to the input feature maps. The kernel size and the dilation size of the temporal convolution kernel for computing the offsets are the same as those of the temporal kernel used for computing output features (e.g., 3×1 with dilation 1 in Figure 4.5). The resulting offset fields have the same size as the input feature map. During training, both the temporal convolutional kernel for generating the output features and the kernel for generating the offsets are learned end-to-end simultaneously.

Dataset	50Salads (mid)			GTEA			JIGSAWS		
Model	F1@{10,25,50}	Edit	Acc	F1@{10,25,50}	Edit	Acc	F1@{10}	Edit	Acc
ED-TCN [65]	68.0,63.9,52.6	59.8	64.7	72.2,69.3,56.0	-	64.0	89.2	84.7	80.8
TUnet [98]	59.3,55.6,44.8	50.6	60.6	67.1,63.7,51.9	60.3	59.9	85.9	79.8	80.2
TResNet [44]	69.2,65.0,54.4	60.5	66.0	74.1,69.9,57.6	64.4	65.8	86.2	85.2	81.1
TDRN	72.9,68.5,57.2	66.0	68.1	79.2,74.4,62.7	74.1	70.1	92.9	90.2	84.6

Table 4.1: Performance comparison with respect to the most related temporal convolution models including ED-TCN [65], TUNet [98] and TResNet [44].

Layer/Module	Kernel Specification
FC	Dense(C, 'softmax')
Conv	Conv1D(64, 50, 1, 1)
DTRM	Conv1D(64, 50, 1, 1, Offsets(96, 50, 1, 1))
DTRM	Conv1D(96, 50, 1, 1, Offsets(64, 50, 1, 1))
DTRM	Conv1D(64, 50, 1, 1, Offsets(64, 50, 1, 1))
Conv	Conv1D(64, 50, 1, 1)

Table 4.2: TDRN architecture: The temporal convolution kernel is described in the same format as in Keras [17], i.e., Conv1D(filters, kernel size, strides, dilation rate). The last argument of a DTRM kernel specifies the temporal convolution kernel corresponding to offsets. C denotes the number of action classes including background class. The fully-connected layer, Dense, is applied to every temporal window of the input video.

4.4 Network Configurations and Training

Our TDRN consists three DTRMs, which are implemented with Keras [17] and TensorFlow. As input, TDRN uses a set of frame-level video features computed by CNNs. We use the same CNN features as in [65]. The output of TDRN is the sequence of action labels assigned to video frames. A detailed description of each module in TDRN is summarized in Table 4.2. For simplicity, Table 4.2 omits the details of NRLUs that follow every deformable temporal convolution layer in DTRM and every temporal convolution layer in TDRN. They are fully specified in Sec. 4.3.2.

Parameters of TDRNs are learned using the categorical cross entropy loss with Stochastic Gradient Descent and ADAM [60] step updates. The batch size and the number of epoches are 8 and 200, respectively. Dropouts are also applied in all the temporal convolutional layers.

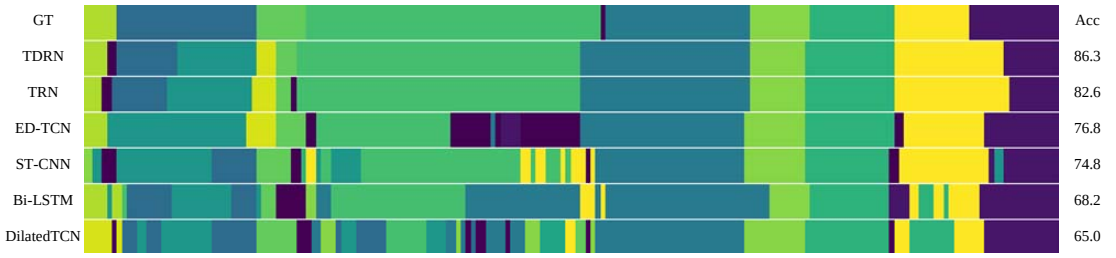


Figure 4.6: Action segmentations for a sample test video named *rgb-22-2.avi* from the 50 Salads dataset. Top-down, the rows correspond to ground truth sequence of actions {place lettuce into bowl, cut cheese, place cheese into bowl, peel cucumber, background, cut cucumber, place cucumber into bowl, mix ingredients, serve salad onto plate, add dressing}, and predictions of TDRN, TRN, ED-TCN [65], ST-CNN [66], Bi-LSTM [106] and Dilated TCN [65].

Note that n does not require careful tuning, because it is not a free parameter, but set to a fixed value that depends on a given network architecture. Specifically, n is uniquely determined by the number of pooling and unpooling operations in the network, as the pooling reduces the temporal length of video processing by a half, and the unpooling doubles the temporal length of video processing. For the model depicted in Figure 4.1 and summarized in Table 4.2, there are 3 residual modules (DTRMs) and 2 pooling/unpooling operations, so the values of n in each of the 3 DTRMs, bottom to top, must be 1, 2 and 1, respectively. We use the same architecture and hence the same values of n for all datasets.

4.5 Experimental Results

We conduct experiments on three challenging action segmentation datasets, including the University of Dundee 50 Salads (50Salads) [110], Georgia Tech Egocentric Activities (GTEA) [29] and the JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) [36]. For evaluation, we use three standard metrics, including F1@k, edit score and accuracy of frame labeling.

4.5.1 Datasets, Metrics and Baselines

Datasets. The 50Salads contains 50 videos with 25 people preparing salad. Each video contains 9000 to 18000 frames with accelerometer data, depth information, RGB data and action label. As input to TDRN, we use the same spatial CNN features as in [65], where the CNN is trained on RGB images showing 17 mid-level action classes. For evaluation on this dataset, we perform the same 5-fold cross-validation as the state of the art, and report the average results.

The GTEA dataset contains 28 videos of seven fine-grained types of daily activities in a kitchen. An activity is performed by four different subjects and each video contains about 1800 RGB frames, showing a sequence of 20 actions including the background action. For fair comparison, input to TDRN are the same CNN features as those used in [65]. For this dataset, we perform the same 4-fold cross-validation as prior work, and report the average results.

The JIGSAWS dataset contains 39 surgical videos with 10 different actions. Each video contains about 3000 frames showing about 20 actions. For fair comparison, we use the same input features of 39 suturing videos as in [68, 65], and perform the same 8-fold cross-validation as prior work, and report the average results.

As in related work [65], we first downsample the video frames and then handle different lengths of downsampled video clips as follows. We first identify the maximum temporal length in the dataset, and then pad zeros to those clips that are shorter than the maximum length.

Metrics. For all the three datasets, we use the following evaluation metrics as in [65]: frame-wise accuracy, segmental edit score, and segmental overlap F1 score with threshold $k/100$, denoted as F1@k. Frame-wise accuracy is one of the most common evaluation metrics for action segmentation. Its drawback is that it does not take into account the temporal structure of the prediction. Consequently, results with large qualitative differences may have the same frame-wise accuracy. Also, this metric does not capture the case of oversegmentation, when the results do not respect the true temporal continuity of human actions, and yet score high frame-wise accuracy. To address these limitations, evaluations presented in [66, 67] additionally use a segmental edit score, which penalizes oversegmentation. The approach of [65] uses F1@k as a suitable metric for testing both action segmentation and action detection, since it also penalizes oversegmentation errors,

50 Salads (mid)	F1@{10,25,50}	Edit	Acc
Spatial CNN [66]	32.3,27.1,18.9	24.8	54.9
IDT+LM [96]	44.4,38.9,27.8	45.8	48.7
Dilated TCN [65]	52.2,47.6,37.4	43.1	59.3
ST-CNN [66]	55.9,49.6,37.1	45.9	59.4
Bi-LSTM [106]	62.6,58.3,47.0	55.6	55.7
ED-TCN [65]	68.0,63.9,52.6	59.8	64.7
TRN	70.2,65.4,56.3	63.7	66.9
TDRN+UNet	69.6,65.0,53.6	62.2	66.1
TDRN	**72.9**,**68.5**,**57.2**	**66.0**	**68.1**

Table 4.3: Results on 50 Salads (mid).

but ignores minor temporal shifts between the predictions and ground truth (which might arise from annotation noise). F1@k score is determined by the total number actions but not depend on the duration of each action instance. It is a metric that similar to mean average precision (mAP) with an Intersection-Over-Union (IoU) overlap criterion which is commonly used in object detection.

Baselines. For ablation studies, we specify the following TDRN variants: (1) TRN : TDRN without deformable convolution (i.e., uses a standard temporal convolution); and (2) TDRN+UNet : TDRN with added TUnet connections, marked cyan in Figure 4.2. We also compare with the following closely related work: (3) Spatial CNN [66]: Frame-wise classification using CNN features of a single RGB frame that capture object texture and spatial location; (4) ST-CNN [66]: Temporal convolutional filter that builds on top of spatial CNN to capture scene changes over the course of action; (5) Bi-LSTM [106]: Bi-directional temporal LSTM; 6) ED-TCN [65]: encoder-decoder temporal convolution neural network; and 7) Dilated TCN [65]: encoder-decoder temporal convolution neural network with dilated temporal convolution. In addition, we compare with the baselines IDT+LM [96], EgoNet+TDD [108], and MSM-CRF [115] and TCN [68] on 50Salads, GTEA and JIGSAWS, respectively.

In our experiments, for fair comparison, we use the same number of temporal convolution layers and kernels as in [65].

Figure 4.7: Action segmentations for a sample test video named *S3-CofHoney-C1.mp4* from the GTEA dataset. Top-down, the rows correspond to ground truth sequence of actions { background, take, background, take, open, background, scoop, pour, background, scoop, pour, background, put, close, background, take, background, open, background, pour, background, put, close, background, take, background, open, background, pour, put, close, background, stir }, and predictions of TDRN, TRN, ED-TCN [65], Bi-LSTM [106], ST-CNN [66] and Dilated TCN [65].

4.5.2 Comparison with Convolution Models

Table 4.1 presents a comparison of TDRN with the most related temporal convolution models, including ED-TCN [65], TUnet [98] and TResNet [44], illustrated in Figure 4.2. The table shows that the performance of TUnet is worse than that of ED-TCN, and that TResNet is able to improve over ED-TCN. Our TDRN outperforms the three related models on all of the datasets. This suggests advantages of explicitly computing the pooling and residual feature streams in TDRN for capturing contextual and fine-scale details, respectively, whereas the three related models do not explicitly compute these streams, as depicted in Figure 4.2.

4.5.3 Comparison with the State of the Art

50Salads. Table 4.3 presents the results of the state of the art, TDRN and its variants. As can be seen, TDRN gives the best performance, since TDRN accounts for multiscale long-range/high-order temporal dependencies as well as frame-level features. Also, TDRN outperforms its variant TRN which uses a standard temporal convolution, suggesting that TDRN is more robust to temporal variations of action boundaries than TRN. We also observe that augmenting TDRN with UNet-like connections in the variant called TDRN+UNet deteriorates performance of TDRN. This is consistent with the re-

GTEA	F1@{10,25,50}	Edit	Acc
Spatial CNN [66]	41.8,36.0,25.1	-	54.1
ST-CNN [66]	58.7,54.4,41.9	-	60.6
Bi-LSTM [106]	66.5,59.0,43.6	-	55.5
Dilated TCN [65]	58.8,52.2,42.2	-	58.3
ED-TCN [65]	72.2,69.3,56.0	-	64.0
EgoNet+TDD [108]	-	-	64.4
TRN	77.4,71.3,59.1	72.2	67.8
TDRN+UNet	78.1,73.8,62.2	73.7	69.3
TDRN	**79.2**,**74.4**,**62.7**	**74.1**	**70.1**

Table 4.4: Results on GTEA.

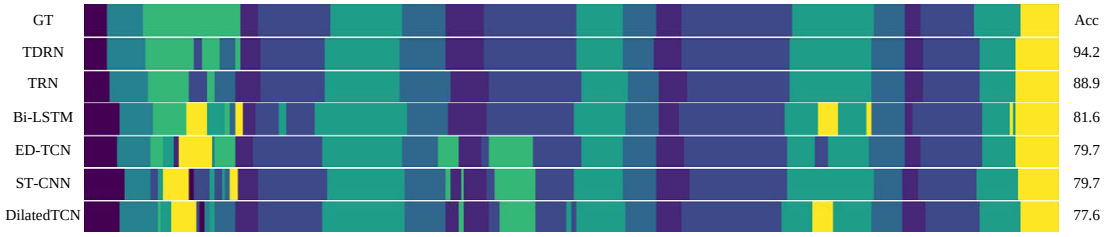


Figure 4.8: Action segmentations for a sample test video named *Suturing-B002.avi* from the JIGSAWS dataset. Top-down, the rows correspond to ground truth sequence of actions in different gestures {G1, G5, G8, G2, G3, G6, G4, G2, G3, G6, G4, G2, G3, G6, G4, G2, G3, G6, G11}, and predictions of TDRN, TRN, Bi-LSTM [106], ED-TCN [65], ST-CNN [66] and Dilated TCN [65].

sults presented in Sec. 4.5.2. Figure 4.6 qualitatively compares our segmentation results with those of the state of the art on a sample test video from the 50 Salads dataset. As can be seen, TDRN does not suffer from oversegmentation. TDRN produces more accurate action recognition than ED-TCN. For example, in Figure 4.6, ED-TCN completely misclassifies the second action in the video, while TDRN is able to generate partially correct prediction. Also, Figure 4.6 shows that TDRN predicts more precise action boundaries. This suggests that using deformable temporal convolution is critical for improving accuracy of prediction of action boundaries.

GTEA. Table 4.4 shows that TDRN and its variants achieve superior segmental overlap F1 score, segmental edit score and frame-wise accuracy than the baselines on the GTEA dataset. Among the state of the art, the best accuracy was achieved by the ap-

JIGSAWS	F1@{10}	Edit	Acc
MSM-CRF [115]	-	-	71.7
Spatial CNN [66]	-	37.7	74.0
ST-CNN [66]	78.3	68.6	78.4
Bi-LSTM [106]	77.8	66.8	77.4
ED-TCN [65]	89.2	84.7	80.8
TCN [68]	-	83.1	81.4
TRN	91.4	87.7	83.3
TDRN+UNet	92.1	89.4	83.9
TDRN	**92.9**	**90.2**	**84.6**

Table 4.5: Results on JIGSAWS.

proach of [108], which combines CNN features with trajectory-pooled deep-convolutional descriptors (TDD) [121]. This suggests that our results could be further improved by incorporating the TDD features. Figure 4.7 qualitatively compares our segmentation results with those of the state of the art on a sample test video from the GTEA dataset.

JIGSAWS. Table 4.5 compares TDRN with the state of the art on the JIGSAWS dataset. Similar to the results on 50Salads and GTEA, TDRN achieves superior performance in all the three metrics. A qualitative comparison on a sample test video from JIGSAWS is depicted in Figure 4.8.

In general, we find that explicit capturing of long-range temporal dependencies by TDRN makes its predictions more reliable than that of the state of the art, especially in cases when distinct actions are visual very similar. For example, in Figure 4.6, ED-TCN wrongly predicts the ground truth class *peel cucumber* as *background* while neglecting the temporal dependencies between consecutive action pair {*peel cucumber, cut cucumber*}. On the other hand, TDRN manages to correctly predict *peel cucumber* most likely because it explicitly accounts for long-range action dependencies. We also find similar examples in the results on the GTEA and JIGSAWS datasets (see Figure 4.7 and Figure 4.8). Empirically, we find that our TDRN sometimes misses the prediction of extremely short action instance that fall in between two long actions, as shown in Figure 4.6 and Figure 4.7.

4.5.4 Effect of Kernel Size and Network Depth

We study performance of TDRN as a function of varying kernel size and network depth, i.e., varying temporal receptive size and number of DTRMs. Figure 4.9 shows F1@10 score of our TDRN on 50Salads (mid). For all network depths, we observe that the score first increases and then drops as the kernel size becomes larger (i.e., when using longer temporal context). This suggests the importance of selecting a suitable kernel size. Our TDRN achieves the best score when the number of DTRMs is 3 (i.e., network depth 6 as specified in Table 4.2) and the kernel size is 50 on the 50Salad dataset. Our optimal network depth agrees with that in [65]. That is, we use the same architecture as specified in Table 4.2 for all datasets for fair comparison with [65].

Note that TDRN training takes 10x less time than for Bi-LSTM, on a Telsa K80 Nvidia gpu card. This is due to independent activations within each temporal convolution layer in TDRN while the activations within Bi-LSTM depend on its previous activations. Hence, for our TDRN, operations can be computed simultaneously in batches.

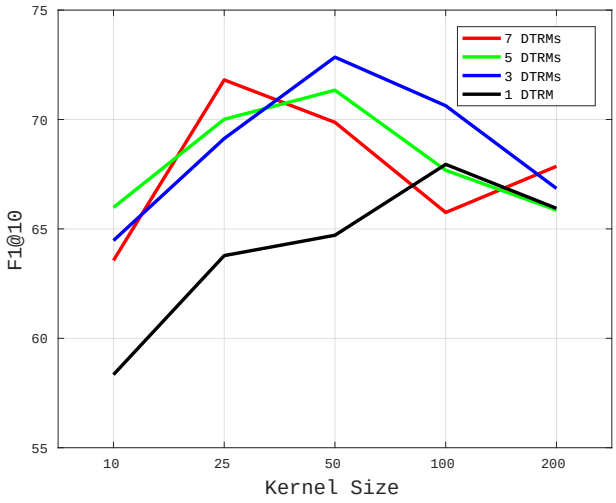


Figure 4.9: F1@10 of TDRN as a fucntion of temporal kernel size and network depth on 50Salads (mid).

4.6 Summary

We have presented a new deep architecture, called temporal deformable convolution neural network (TDRN), for action segmentation in videos. TDRN consists of two processing streams: a temporal pooling stream that captures long-range and high-level features at multiple temporal scales, and a temporal residual stream that computes features at the same frame-level temporal resolution as the input video. As the pooling stream accounts for temporal context, it is aimed at improving action recognition. The residual stream is aimed at improving localization of action boundaries. The two streams are aggregated by a cascade of deformable temporal residual modules, each computing deformable temporal convolutions for modeling temporal variations in action boundaries.

Our empirical evaluation on the benchmark University of Dundee 50 Salads (50Salads), Georgia Tech Egocentric Activities (GTEA) and JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) demonstrates that TDRN outperforms the state-of-the-art convolution and temporal convolution models. TDRN produces more accurate action boundary detections, which suggest advantages of our end-to-end learning of deformable temporal convolution over using the standard temporal convolution. Also, TDRN's results tend to better respect common-sense temporal arrangement of actions, due to its explicit learning of long-range temporal dependencies. We have empirically found that TDRN sometimes poorly segments very short actions that fall in between two long actions.

Chapter 5: Conclusion

In this dissertation, we have addressed the basic vision problem that of robust video labeling at both the frame and pixel levels. For pixel-level video labeling, we have addressed two problems: spatiotemporal video segmentaton and boundary flow estimation (i.e., joint object boundary pixels labeling and boundary motion estimation). For frame-level video labeling, we have addressed temporal action segmentation.

We advance the state of the art by designing novel deep architectures that are driven by our two hypotheses: (i) Robust and accurate video labeling requires accounting for both local and long-range spatial and temporal cues; and (ii) Accuracy of pixel-level video labeling improves by modeling and inferring of object boundaries, and similarly for the frame-level video labeling, temporal action boundaries.

For spatiotemporal video segmentation, we have presented recurrent temporal deep field (RTDF) which captures long-range and high-order spatiotemporal dependencies of pixel labels in a video by combining conditional random field (CRF), deconvolution neural network (DeconvNet), and recurrent temporal restricted Boltzmann machine (RTRBM) into a unified framework. CRF is suitable for modeling local interactions while RTRBM is appropriate for modeling global properties of object shapes. Our empirical evaluation on the benchmark Youtube Face Database (YFDB) [125] and Cambridge-driving Labeled Video Database (CamVid) [9] has demonstrated the advantages of performing joint inference and joint training of RTDF, resulting in its superior performance over the state of the art.

For boundary flow estimation, we have introduced a rigorous definition of boundary flow and specified a new end-to-end trainable fully convolutional Siamese network (FCSN). FCSN unifies boundary detection and boundary flow estimation. FCSN takes two images as input and produces boundary detections in each image. FCSN also generates excitation attention maps in the two images as informative features for boundary matching. Our experiments on the benchmark VSB100 dataset [33] for boundary detection have demonstrated that FCSN is superior to the state of the art, successfully detecting boundaries both of foreground and background objects. We have presented

the first results of boundary flow on the benchmark Sintel training set [12] and demonstrated the utility of boundary flow by integrating our approach with the CPM-Flow [49] for dense optical flow estimation, resulting in an improved performance over the original CPM-Flow.

For temporal action segmentation in videos, we have presented the temporal deformable convolution neural network (TDRN). TDRN integrates local, fine-scale cues and global, long-range video information via two processing streams – a temporal pooling stream aimed at improving action recognition by accounting for temporal context and a temporal residual stream aimed at improving localization of action boundaries by using fine-scale temporal cues. The two streams are aggregated by a cascade of deformable temporal residual modules (DTRMs), each computing deformable temporal convolutions for modeling temporal variations in action boundaries. Our empirical evaluation on the benchmark University of Dundee 50 Salads (50Salads) [110], Georgia Tech Egocentric Activities (GTEA) [29] and JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) [36] has demonstrated that TDRN outperforms the state-of-the-art temporal convolution models. TDRN produces more accurate action boundary detections. This supports our hypotheses that end-to-end learning of deformable temporal convolution is more advantageous than the standard temporal convolution. Also, TDRNs results tend to better respect temporal arrangement of actions, due to our explicit learning of long-range temporal dependencies.

In summary, our empirical evaluations on the benchmark datasets have demonstrated validity of our hypotheses that robust and accurate video labeling requires accounting for both local and long-range spatial and temporal cues as well as object/action boundaries.

Bibliography

[1] Mohamed R Amer, Siavash Yousefi, Raviv Raich, and Sinisa Todorovic. Monocular extraction of 2.1 d sketch using constrained convex optimization. *International Journal of Computer Vision*, 112(1):23–42, 2015.

[2] Mohamed Rabie Amer, Peng Lei, and Sinisa Todorovic. Hirf: Hierarchical random field for collective activity recognition in videos. In *ECCV*, pages 572–585. Springer, 2014.

[3] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2011.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[5] Christian Bailer, Bertram Taetz, and Didier Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. In *ICCV*, 2015.

[6] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009.

[7] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection. In *CVPR*, 2015.

[8] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. High-for-low and low-for-high: Efficient boundary detection from deep object features and its applications to high-level vision. In *ICCV*, 2015.

[9] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *PRL*, 2008.

[10] Matthew Brown and David G Lowe. Recognising panoramas. In *ICCV*, 2003.

[11] Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In *ECCV*, 2010.

[12] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012.

[13] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2014.

[14] Qifeng Chen and Vladlen Koltun. Full flow: Optical flow estimation by global optimization over regular grids. In *CVPR*, 2016.

[15] Sheng Chen, Alan Fern, and Sinisa Todorovic. Multi-object tracking via constrained sequential labeling. In *CVPR*, pages 1130–1137, 2014.

[16] Yuhua Chen, Jordi Pont-Tuset, Alberto Montes, and Luc Van Gool. Blazingly fast video object segmentation with pixel-wise metric learning. In *CVPR*, 2018.

[17] François Chollet. Keras (2015). *http://keras.io*, 2017.

[18] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, 2012.

[19] Ionut Cosmin Duta, Bogdan Ionescu, Kiyoharu Aizawa, and Nicu Sebe. Spatio-temporal vector of locally max pooled features for action recognition in videos. In *CVPR*, pages 3097–3106, 2017.

[20] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017.

[21] Xiyang Dai, Bharat Singh, Guyue Zhang, Larry S Davis, and Yan Qiu Chen. Temporal context network for activity localization in videos. In *ICCV*, 2017.

[22] Achal Dave, Olga Russakovsky, and Deva Ramanan. Predictive-corrective networks for action detection. In *CVPR*, 2017.

[23] Li Ding and Chenliang Xu. Tricornet: A hybrid temporal convolutional and recurrent network for video action segmentation. *arXiv preprint arXiv:1705.07818*, 2017.

[24] Piotr Dollár and C Lawrence Zitnick. Fast edge detection using structured forests. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1558–1570, 2015.

[25] SM Ali Eslami, Nicolas Heess, Christopher KI Williams, and John Winn. The shape boltzmann machine: a strong model of object shape. *IJCV*, 107(2):155–176, 2014.

[26] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *PAMI*, 35(8):1915–1929, 2013.

[27] Alireza Fathi, Ali Farhadi, and James M Rehg. Understanding egocentric activities. In *ICCV*, pages 407–414. IEEE, 2011.

[28] Alireza Fathi and James M Rehg. Modeling actions through state changes. In *CVPR*, pages 2579–2586, 2013.

[29] Alireza Fathi, Xiaofeng Ren, and James M Rehg. Learning to recognize objects in egocentric activities. In *CVPR*, pages 3281–3288. IEEE, 2011.

[30] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, pages 3468–3476, 2016.

[31] Christoph Feichtenhofer, Axel Pinz, and Richard P Wildes. Spatiotemporal multiplier networks for video action recognition. In *CVPR*, pages 4768–4777, 2017.

[32] Vittorio Ferrari, Loic Fevrier, Frederic Jurie, and Cordelia Schmid. Groups of adjacent contour segments for object detection. *IEEE transactions on pattern analysis and machine intelligence*, 30(1):36–51, 2008.

[33] F. Galasso, N.S. Nagaraja, T.J. Cardenas, T. Brox, and B.Schiele. A unified video segmentation benchmark: Annotation, metrics and analysis. In *ICCV*, 2013.

[34] Eric Galmar, Th Athanasiadis, Benoit Huet, and Y Avrithis. Spatiotemporal semantic video segmentation. In *MSPW*, 2008.

[35] Yaroslav Ganin and Victor Lempitsky. Nˆ4-fields: Neural network nearest neighbor fields for image transforms. In *ACCV*, 2014.

[36] Yixin Gao, S Swaroop Vedula, Carol E Reiley, Narges Ahmidi, Balakrishnan Varadarajan, Henry C Lin, Lingling Tao, Luca Zappella, Benjamın Béjar, David D Yuh, et al. Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling. In *MICCAI Workshop*, volume 3, 2014.

[37] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[38] Rohit Girdhar, Deva Ramanan, Abhinav Gupta, Josef Sivic, and Bryan Russell. Actionvlad: Learning spatio-temporal aggregation for action classification. In *CVPR*, 2017.

[39] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Bidirectional lstm networks for improved phoneme classification and recognition. *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*, pages 753–753, 2005.

[40] Matthias Grundmann, Vivek Kwatra, Mei Han, and Irfan Essa. Efficient hierarchical graph-based video segmentation. In *CVPR*, 2010.

[41] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *ECCV*, 2014.

[42] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *ECCV*, 2014.

[43] Kaiming He and Jian Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *CVPR*, 2012.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[45] Xuming He, Richard S Zemel, and Miguel Á Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *CVPR*, 2004.

[46] G. E Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 2002.

[47] Rui Hou, Chen Chen, and Mubarak Shah. An end-to-end 3d convolutional neural network for action detection and segmentation in videos. *arXiv preprint arXiv:1712.01111*, 2017.

[48] Rui Hou, Chen Chen, and Mubarak Shah. Tube convolutional neural network (t-cnn) for action detection in videos. In *ICCV*, 2017.

[49] Yinlin Hu, Rui Song, and Yunsong Li. Efficient coarse-to-fine patchmatch for large displacement optical flow. In *CVPR*, 2016.

[50] De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. Connectionist temporal modeling for weakly supervised action labeling. In *ECCV*, pages 137–153. Springer, 2016.

[51] Ahmad Humayun, Fuxin Li, and James M Rehg. The middle child problem: Revisiting parametric min-cut and seeds for object proposals. In *ICCV*, 2015.

[52] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[53] Abhishek Jain, Saptarshi Chatterjee, and René Vidal. Coarse-to-fine semantic video segmentation using supervoxel trees. In *ICCV*, 2013.

[54] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[55] Andrew Kae, Benjamin Marlin, and Erik Learned-Miller. The shape-time random field for semantic video labeling. In *CVPR*, 2014.

[56] Andrew Kae, Kihyuk Sohn, Honglak Lee, and Erik Learned-Miller. Augmenting crfs with boltzmann machine shape priors for image labeling. In *CVPR*, 2013.

[57] Vicky Kalogeiton, Philippe Weinzaepfel, Vittorio Ferrari, and Cordelia Schmid. Action tubelet detector for spatio-temporal action localization. In *ICCV*, 2017.

[58] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, pages 1725–1732, 2014.

[59] Anna Khoreva, Rodrigo Benenson, Fabio Galasso, Matthias Hein, and Bernt Schiele. Improved image boundaries for better video segmentation. In *ECCV 2016 Workshops*, 2016.

[60] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[61] Simon Korman and Shai Avidan. Coherency sensitive hashing. In *ICCV*, 2011.

[62] Hilde Kuehne, Juergen Gall, and Thomas Serre. An end-to-end generative framework for video segmentation and recognition. In *WACV*, pages 1–8. IEEE, 2016.

[63] L'ubor Ladickỳ, Paul Sturgess, Karteek Alahari, Chris Russell, and Philip HS Torr. What, where and how many? combining object detectors and CRFs. In *ECCV*, 2010.

[64] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[65] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *CVPR*, 2017.

[66] Colin Lea, Austin Reiter, René Vidal, and Gregory D Hager. Segmental spatiotemporal cnns for fine-grained action segmentation. In *ECCV*, pages 36–52. Springer, 2016.

[67] Colin Lea, René Vidal, and Gregory D Hager. Learning convolutional action primitives for fine-grained action recognition. In *ICRA*, pages 1642–1649. IEEE, 2016.

[68] Colin Lea, René Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks: A unified approach to action segmentation. In *ECCV Workshops*, pages 47–54. Springer, 2016.

[69] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1:0, 2006.

[70] Yong Jae Lee, Jaechul Kim, and Kristen Grauman. Key-segments for video object segmentation. In *ICCV*, 2011.

[71] Peng Lei, Fuxin Li, and Sinisa Todorovic. Boundary flow: A siamese network that predicts boundary motion without training on motion. In *CVPR*, pages 3282–3290, 2018.

[72] Peng Lei and Sinisa Todorovic. Modeling human-skeleton motion patterns using conditional deep boltzmann machine. In *ICPR*, pages 1845–1850. IEEE, 2016.

[73] Peng Lei and Sinisa Todorovic. Recurrent temporal deep field for semantic video labeling. In *ECCV*, pages 302–317. Springer, 2016.

[74] Peng Lei and Sinisa Todorovic. Temporal deformable residual networks for action segmentation in videos. In *CVPR*, pages 6742–6751, 2018.

[75] Peng Lei, Tianfu Wu, Mingtao Pei, Anlong Ming, and Zhenyu Yao. Robust tracking by accounting for hard negatives explicitly. In *ICPR*, pages 2112–2115. IEEE, 2012.

[76] Fuxin Li, Taeyoung Kim, Ahmad Humayun, David Tsai, and James M Rehg. Video segmentation by tracking many figure-ground segments. In *ICCV*, 2013.

[77] Yanghao Li, Cuiling Lan, Junliang Xing, Wenjun Zeng, Chunfeng Yuan, and Jiaying Liu. Online human action detection using joint classification-regression recurrent neural networks. In *ECCV*, pages 203–220. Springer, 2016.

[78] Yujia Li, Daniel Tarlow, and Richard Zemel. Exploring compositional high order pattern potentials for structured output learning. In *CVPR*, 2013.

[79] Tianwei Lin, Xu Zhao, and Zheng Shou. Single shot temporal action detection. In *MM*, 2017.

[80] Buyu Liu, Xuming He, and Stephen Gould. Multi-class semantic video segmentation with exemplar-based object reasoning. In *WACV*, 2015.

[81] Ce Liu, William T Freeman, and Edward H Adelson. Analysis of contour motions. In *NIPS*, pages 913–920, 2006.

[82] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[83] Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Pablo Arbeláez, and Luc Van Gool. Convolutional oriented boundaries. In *ECCV*, 2016.

[84] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016.

[85] Moritz Menze, Christian Heipke, and Andreas Geiger. Discrete optimization for optical flow. In *GCPR*, 2015.

[86] Volodymyr Mnih, Hugo Larochelle, and Geoffrey E Hinton. Conditional restricted boltzmann machines for structured output prediction. In *UAI*, 2011.

[87] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[88] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015.

[89] Teddy Ort, Liam Paull, and Daniela Rus. Autonomous vehicle navigation in rural environments without detailed prior maps. In *ICRA*, 2018.

[90] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318, 2013.

[91] Xiaojiang Peng and Cordelia Schmid. Multi-region two-stream r-cnn for action detection. In *ECCV*, pages 744–759. Springer, 2016.

[92] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene parsing. In *ICML*, 2014.

[93] Tobias Pohlen, Alexander Hermans, Markus Mathias, and Bastian Leibe. Full-resolution residual networks for semantic segmentation in street scenes. In *CVPR*, 2017.

[94] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.

[95] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *CVPR*, 2015.

[96] Alexander Richard and Juergen Gall. Temporal action detection using a statistical language model. In *CVPR*, pages 3131–3140, 2016.

[97] Alexander Richard, Hilde Kuehne, and Juergen Gall. Weakly supervised action learning with rnn based fine-to-coarse modeling. In *CVPR*, 2017.

[98] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–241. Springer, 2015.

[99] S Rota Bulo and Peter Kontschieder. Neural decision forests for semantic image labelling. In *CVPR*, 2014.

[100] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[101] Amirreza Shaban, Alrik Firl, Ahmad Humayun, Jialin Yuan, Xinyao Wang, Peng Lei, Nikhil Dhanda, Byron Boots, James M Rehg, and Fuxin Li. Multiple-instance video segmentation with sequence-specific object proposals. In *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*, 2017.

[102] Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, and Zhijiang Zhang. Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *CVPR*, 2015.

[103] Zheng Shou, Jonathan Chan, Alireza Zareian, Kazuyuki Miyazawa, and Shih-Fu Chang. Cdc: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos. In *CVPR*, 2017.

[104] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, pages 568–576, 2014.

[105] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[106] Bharat Singh, Tim K Marks, Michael Jones, Oncel Tuzel, and Ming Shao. A multi-stream bi-directional recurrent neural network for fine-grained action detection. In *CVPR*, pages 1961–1970, 2016.

[107] Gurkirt Singh, Suman Saha, and Fabio Cuzzolin. Online real time multiple spatiotemporal action localisation and prediction on a single platform. In *ICCV*, 2017.

[108] Suriya Singh, Chetan Arora, and CV Jawahar. First person action recognition using deep learned descriptors. In *CVPR*, pages 2620–2628, 2016.

[109] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*. MIT Press Cambridge, 1986.

[110] Sebastian Stein and Stephen J McKenna. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *Ubicomp*, pages 729–738. ACM, 2013.

[111] Paul Sturgess, Karteek Alahari, Lubor Ladicky, and Philip HS Torr. Combining appearance and structure from motion features for road scene understanding. In *BMVC*, 2009.

[112] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *CVPR*, 2018.

[113] Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *ICCV*, pages 4597–4605, 2015.

[114] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *NIPS*, 2009.

[115] Lingling Tao, Luca Zappella, Gregory D Hager, and René Vidal. Surgical gesture segmentation and recognition. In *MICCAI*, pages 339–346. Springer, 2013.

[116] Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. Modeling human motion using binary latent variables. In *NIPS*, 2006.

[117] William B Thompson. Exploiting discontinuities in optical flow. *International Journal of Computer Vision*, 30(3):163–173, 1998.

[118] William B Thompson, Kathleen M Mutch, and Valdis A Berzins. Dynamic occlusion analysis in optical flow fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (4):374–383, 1985.

[119] Joseph Tighe and Svetlana Lazebnik. Superparsing. *IJCV*, 101(2):329–349, 2013.

[120] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, pages 4489–4497, 2015.

[121] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *CVPR*, pages 4305–4314, 2015.

[122] Limin Wang, Yuanjun Xiong, Dahua Lin, and Luc Van Gool. Untrimmednets for weakly supervised action recognition and detection. In *CVPR*, pages 4325–4334, 2017.

[123] Yunbo Wang, Mingsheng Long, Jianmin Wang, and Philip S Yu. Spatiotemporal pyramid network for video action recognition. In *CVPR*, pages 1529–1538, 2017.

[124] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *ICCV*, 2013.

[125] Lior Wolf, Tal Hassner, and Itay Maoz. Face recognition in unconstrained videos with matched background similarity. In *CVPR*, 2011.

[126] Zhengyang Wu, Fuxin Li, Rahul Sukthankar, and James M Rehg. Robust video segment proposals with painless occlusion handling. In *CVPR*, 2015.

[127] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015.

[128] Huijuan Xu, Abir Das, and Kate Saenko. R-c3d: Region convolutional 3d network for temporal activity detection. In *ICCV*, 2017.

[129] Tianfan Xue, Michael Rubinstein, Ce Liu, and William T Freeman. A computational approach for obstruction-free photography. *ACM Transactions on Graphics (TOG)*, 34(4):79, 2015.

[130] Jimei Yang, Brian Price, Scott Cohen, Honglak Lee, and Ming-Hsuan Yang. Object contour detection with a fully convolutional encoder-decoder network. In *CVPR*, 2016.

[131] Serena Yeung, Olga Russakovsky, Ning Jin, Mykhaylo Andriluka, Greg Mori, and Li Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. *International Journal of Computer Vision*, pages 1–15, 2015.

[132] Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *CVPR*, pages 2678–2687, 2016.

[133] Saehoon Yi and Vladimir Pavlovic. Multi-cue structure preserving mrf for unconstrained video segmentation. *arXiv preprint arXiv:1506.09124*, 2015.

[134] Zehuan Yuan, Jonathan C Stroud, Tong Lu, and Jia Deng. Temporal action localization by structured maximal sums. In *CVPR*, 2017.

[135] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, pages 4694–4702, 2015.

[136] Chenxi Zhang, Liang Wang, and Ruigang Yang. Semantic segmentation of urban scenes using dense depth maps. In *ECCV*, 2010.

[137] Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. In *ECCV*, 2016.

[138] Handong Zhao and Yun Fu. Semantic single video segmentation with robust graph representation. In *IJCAI*, 2015.

[139] Yue Zhao, Yuanjun Xiong, Limin Wang, Zhirong Wu, Dahua Lin, and Xiaoou Tang. Temporal action detection with structured segment networks. In *ICCV*, 2017.

[140] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015.

[141] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, 2014.

APPENDICES

Appendix A: Derivation of Mean-field Updating Equations

In the following we present a detailed derivation of the mean-field inference algorithm which is explained in Section 2.4 in Chapter 2 (i.e., Inference of RTDF). Here, we use the same notation as in Chapter 2.

The Kullback-Leibler divergence between $Q(\mathbf{y}^t, \mathbf{h}^t; \boldsymbol{\mu}, \boldsymbol{\nu})$ and $P(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t)$ is defined as

$$\begin{aligned} KL(Q||P) &= \sum_{\mathbf{y}^t, \mathbf{h}^t} Q(\mathbf{y}^t, \mathbf{h}^t; \boldsymbol{\mu}, \boldsymbol{\nu}) \ln \frac{Q(\mathbf{y}^t, \mathbf{h}^t; \boldsymbol{\mu}, \boldsymbol{\nu})}{P(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t)} \\ &= -H(Q) - \sum_{\mathbf{y}^t, \mathbf{h}^t} Q(\mathbf{y}^t, \mathbf{h}^t; \boldsymbol{\mu}, \boldsymbol{\nu}) \ln P(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t) \end{aligned} \tag{A.1}$$

where $H(Q)$ is the entropy of Q. It follows that minimizing the KL-divergence amounts to the following objective

$$\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}} = \arg\max_{\boldsymbol{\mu}, \boldsymbol{\nu}} \{ \sum_{\mathbf{y}^t, \mathbf{h}^t} Q(\boldsymbol{\mu}, \boldsymbol{\nu}) \ln P(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t) + H(Q(\boldsymbol{\mu}, \boldsymbol{\nu})) \}. \tag{A.2}$$

Substitute $P(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t) = \frac{1}{Z(\theta)} \exp(-E_{\text{RTDF}}(\mathbf{y}^t, \mathbf{h}^t | \mathbf{y}^{<t}, \mathbf{I}^t))$ and $Q(\mathbf{y}^t, \mathbf{h}^t) = Q(\boldsymbol{\mu}, \boldsymbol{\nu}) = \prod_p \mu_p \prod_j \nu_j$ into (A.2). Note that $\boldsymbol{\mu} = \{\mu_{pl} : \mu_{pl} = Q(\mathbf{y}^t_{pl} = 1)\}$, where $\sum_{l=1}^{L} \mu_{pl} = 1$ for every pixel p, $\boldsymbol{\nu} = \{\nu_j : \nu_j = Q(\mathbf{h}^t_j = 1)\}$ and $Z(\theta)$ is the partition function. The

objective takes a particular form:

$$\begin{aligned}\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}} = \arg\max_{\boldsymbol{\mu},\boldsymbol{\nu}}\{ & \sum_p \sum_{l=1}^{L} W^1_{\mu_{pl}} \cdot \mathbf{x}_p + \sum_p \sum_{l=1}^{L} \beta_{p' \to p} \\ + & \sum_i \sum_j \sum_{p \in i} \sum_{l=1}^{L} \frac{1}{|i|} W_{ijl} \nu_j \mu_{pl} + \sum_j \sum_k W'_{jk} \nu_j r_k \\ + & \sum_i \sum_{l=1}^{L} \sum_{p \in i} \frac{1}{|i|} \mu_{pl} c_{il} + \sum_j b_j \nu_j - \ln Z(\theta) \\ - & \sum_j \{\nu_j \ln \nu_j + (1-\nu_j)\ln(1-\nu_j)\} - \sum_p \sum_{l=1}^{L} \mu_{pl} \ln \mu_{pl}. \quad \text{(A.3)}\end{aligned}$$

We can omit the $\ln Z(\theta)$ term in (A.3) since $Z(\theta)$ is not a function of $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$. Take the derivative with respect to ν_j and set it to 0, we can get that $\ln \frac{\nu_j}{1-\nu_j} = b_j + \sum_i \sum_l \sum_{p \in i} \frac{1}{|i|} W_{ijl} \mu_{pl} + \sum_k W'_{jk} r_k$. It follows that

$$\nu_j = \sigma(\sum_i \sum_l \sum_{p \in i} \frac{1}{|i|} W_{ijl} \mu_{pl} + \sum_k W'_{jk} r_k + b_j). \quad \text{(A.4)}$$

Similarly, for μ_{pl}, we can get that $\ln \mu_{pl} = W^1_{\mu_{pl}} \cdot \mathbf{x}_p + \beta_{p' \to p} + \sum_j W_{ikl} \nu_j + c_{il} - 1$ where $\sum_{l=1}^{L} \mu_{pl} = 1$. For clarity in presentation, we omit the bias term c_{il} in μ_{pl}. It follows that

$$\mu_{pl} = \frac{\exp(W^1_{\mu_{pl}} \cdot \mathbf{x}_p + \sum_j W_{ijl} \nu_j + \beta_{p' \to p})}{\sum_{l'} \exp(W^1_{\mu_{pl'}} \cdot \mathbf{x}_p + \sum_j W_{ijl'} \nu_j + \beta_{p' \to p})}. \quad \text{(A.5)}$$

Note that the first term in (A.4) and the second term in (A.5) use the deterministic mapping between patches i and pixels $p \in i$, as specified in Section 2.3.1 in Chapter 2 (i.e., A Brief Review of Restricted Boltzmann Machine).

Appendix B: Derivation of the Joint Training of RTDF Parameters

In the following we present a detailed derivation of the joint learning algorithm, which is explained in Section 2.5 in Chapter 2 (i.e., Learning). Here, we use the same notation as in Chapter 2.

The goal of joint training is to maximize the conditional log-likelihood $\sum_t \log p(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t)$. We use CD-PercLoss algorithm [86] and error back-propagation (EBP) to jointly train parameters of RTDF in an end-to-end fashion. In this chapter, we use generalized perceptron loss [69]. CD-PercLoss with generalized perceptron loss directly penalizes the model for wrong predictions during training, which makes it suitable for structured output prediction problems. The training objective is to minimize the following generalized perceptron loss with regularization:

$$\triangle(\theta) = \sum_t (F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t) - \min_{\hat{\mathbf{y}}^t} F(\hat{\mathbf{y}}^t|\mathbf{y}^{<t}, \mathbf{I}^t)) + \lambda\theta^T\theta \tag{B.1}$$

where $\theta = \{W^1, W^2, \theta_{\text{DN}}, \theta_{\text{RTRBM}}\}$. It is impossible to get an analytical solution. We use gradient descent algorithm, which leads to the following update rule:

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \eta \frac{\partial \triangle(\theta^{(\tau)})}{\partial \theta^{(\tau)}}. \tag{B.2}$$

In (B.2), η is the learning rate. For calculating the the gradient of the generalized perceptron loss, we have to obtain $\frac{\partial F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t)}{\partial \theta}$ and $\frac{\partial F(\hat{\mathbf{y}}^t|\mathbf{y}^{<t}, \mathbf{I}^t)}{\partial \theta}$ where $F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t)$ denotes the free energy of ground truth label $\mathbf{y}^t$ of frame t, and $\hat{\mathbf{y}}^t$ is the predicted label associated with minimum free energy. In the following, we specify $\frac{\partial F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t)}{\partial \theta}$. The calculation of $\frac{\partial F(\hat{\mathbf{y}}^t|\mathbf{y}^{<t}, \mathbf{I}^t)}{\partial \theta}$ is similar to $\frac{\partial F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t)}{\partial \theta}$.

The free energy of RTDF associated with $\mathbf{y}^t$ is defined as

$$\begin{aligned} F(\mathbf{y}^t|\mathbf{y}^{<t}, \mathbf{I}^t) &= -\sum_p \psi_1(\mathbf{x}_p^t, \mathbf{y}_p^t) - \sum_{p,p'} \psi_2(\mathbf{y}_p^t, \mathbf{y}_{p'}^t) + F_{\text{RTRBM}}(\mathbf{y}^t|\mathbf{r}^{t-1}) \\ &= -\sum_p \psi_1(\mathbf{x}_p^t, \mathbf{y}_p^t) - \sum_{p,p'} \psi_2(\mathbf{y}_p^t, \mathbf{y}_{p'}^t) + F_{\text{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1}). \end{aligned} \tag{B.3}$$

In (B.3), the associated free energy of $\mathbf{z}^t$ given $\mathbf{r}^{t-1}$ is defined as

$$\begin{aligned} F_{\text{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1}) &= -\sum_j \log(1+\exp(b_j+\sum_{i,l} W_{ijl} z_{il}^t + \sum_k W'_{jk} r_k^{t-1})) \\ &\quad -\sum_{i,l} z_{il}^t c_{il} \end{aligned} \tag{B.4}$$

where r_k^{t-1} is a function of $\mathbf{z}^{t-1}$ and $\mathbf{r}^{t-2}$ parametrized by $\mathbf{b}, W$ and W', as specified in Section 2.3.2 in Chapter 2 (i.e., A Brief Review of RTRBM).

It is straightforward to find the derivatives w.r.t. CRF parameters W_1, W_2, so we omit this derivation. Similarly, it is straightforward to find the derivatives w.r.t. bias terms of RTRBM, i.e., $\{\mathbf{b}_{\text{int}}, \mathbf{b}, \mathbf{c}\}$, so we omit this derivation.

Regarding the gradient with respect to θ_{DN}, we first obtain $\frac{\partial F(\mathbf{y}^t|\mathbf{y}^{<t},\mathbf{I}^t)}{\partial \mathbf{x}_p}$ and then back-propagate the gradient using chain rule, i.e., $\frac{\partial F(\mathbf{y}^t|\mathbf{y}^{<t},\mathbf{I}^t)}{\partial \mathbf{x}_p}\frac{\partial \mathbf{x}_p}{\partial \theta_{\text{DN}}}$.

In the following, we specify the gradient with respect to W and W'. The gradient with respect to W_{ijl} is given by

$$\begin{aligned} &\frac{\partial F(\mathbf{y}^t|\mathbf{y}^{<t},\mathbf{I}^t)}{\partial W_{ijl}} = \frac{\partial F_{\text{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1})}{\partial W_{ijl}} \\ = & -\sum_m \frac{\partial F_{\text{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1})}{\partial r_m^{t-1}}\frac{\partial r_m^{t-1}}{\partial W_{ijl}} - z_{il}^t \frac{\exp(b_j+\sum_{i,l} W_{ijl} z_{il}^t + \sum_k W'_{jk} r_k^{t-1})}{1+\exp(b_j+\sum_{i,l} W_{ijl} z_{il}^t + \sum_k W'_{jk} r_k^{t-1})} \\ = & -\sum_m \frac{\partial F_{\text{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1})}{\partial r_m^{t-1}}\frac{\partial r_m^{t-1}}{\partial W_{ijl}} - z_{il}^t \sigma(b_j+\sum_{i,l} W_{ijl} z_{il}^t + \sum_k W'_{jk} r_k^{t-1}) \\ = & -\sum_{j'}[\sigma(b_{j'}+\sum_{i,l} W_{ij'l} z_{il}^t + \sum_k W'_{j'k} r_k^{t-1}) \sum_m W'_{j'm}\frac{\partial r_m^{t-1}}{\partial W_{ijl}}] \\ & -z_{il}^t \sigma(b_j+\sum_{i,l} W_{ijl} z_{il}^t + \sum_k W'_{jk} r_k^{t-1}) \end{aligned} \tag{B.5}$$

where $\frac{\partial r_m^{t-1}}{\partial W_{ijl}} = r_m^{t-1}(1-r_m^{t-1})[z_{il}^{t-1}1(m=j) + \sum_{k'} W'_{mk'}\frac{\partial r_{k'}^{t-2}}{\partial W_{ijl}}]$, $r_m^{t-1} = \sigma(\sum_{i,l} W_{iml}\ z_{il}^{t-1} + \sum_{k'} W'_{mk'} r_{k'}^{t-2} + b_m)$, $\sigma(\cdot)$ denotes sigmoid function and $1(\cdot)$ represents indicator function.

Similarly, we can obtain the gradient with respect to $W^{'}_{jk}$, which is as follows:

$$
\begin{aligned}
&\frac{\partial F(\mathbf{y}^t|\mathbf{y}^{<t},\mathbf{I}^t)}{\partial W^{'}_{jk}} = \frac{\partial F_{\mathrm{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1})}{\partial W^{'}_{jk}} \\
= \; & -\sum_m \frac{\partial F_{\mathrm{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1})}{\partial r^{t-1}_m}\frac{\partial r^{t-1}_m}{\partial W^{'}_{jk}} - r^{t-1}_k \frac{\exp(b_j+\sum_{i,l} W_{ijl}z^t_{il}+\sum_{k'} W^{'}_{jk'}r^{t-1}_{k'})}{1+\exp(b_j+\sum_{i,l} W_{ijl}z^t_{il}+\sum_{k'} W^{'}_{jk'}r^{t-1}_{k'})} \\
= \; & -\sum_m \frac{\partial F_{\mathrm{RTRBM}}(\mathbf{z}^t|\mathbf{r}^{t-1})}{\partial r^{t-1}_m}\frac{\partial r^{t-1}_m}{\partial W^{'}_{jk}} - r^{t-1}_k \sigma(b_j+\sum_{i,l} W_{ijl}z^t_{il}+\sum_{k'} W^{'}_{jk'}r^{t-1}_{k'}) \\
= \; & -[\sum_{j'} \sigma(b_{j'}+\sum_{i,l} W_{ij'l}z^t_{il}+\sum_{k'} W^{'}_{j'k'}r^{t-1}_{k'}) \sum_m W^{'}_{j'm}\frac{\partial r^{t-1}_m}{\partial W^{'}_{jk}}] \\
& -r^{t-1}_k \sigma(b_j+\sum_{i,l} W_{ijl}z^t_{il}+\sum_{k'} W^{'}_{jk'}r^{t-1}_{k'})
\end{aligned} \tag{B.6}
$$

where $\frac{\partial r^{t-1}_m}{\partial W^{'}_{jk}} = r^{t-1}_m(1-r^{t-1}_m)[r^{t-2}_k 1(m=j) + \sum_{k'} W^{'}_{mk'}\frac{\partial r^{t-2}_{k'}}{\partial W^{'}_{jk}}]$, $r^{t-1}_m = \sigma(\sum_{i,l} W_{iml}$ $z^{t-1}_{il} + \sum_{k'} W^{'}_{mk'}r^{t-2}_{k'} + b_m)$, $\sigma(\cdot)$ denotes sigmoid function and $1(\cdot)$ represents indicator function. Note that $\mathbf{r}^{t-\gamma} = \sigma(W\mathbf{z}^{t-\gamma} + \mathbf{b} + \mathbf{b}_{int})$, which indicates that $\frac{\partial r^{t-\gamma}_m}{\partial W_{ijl}} = r^{t-\gamma}_m(1-r^{t-\gamma}_m)z^{t-\gamma}_{il}1(m=j)$ and $\frac{\partial r^{t-\gamma}_m}{\partial W^{'}_{jk}} = 0$.