Spatiotemporal Human Action Detection and Instance Segmentation in Videos

Suman Saha

Artificial Intelligence and Vision Group
School of Engineering, Computing and Mathematics
Oxford Brookes University

A dissertation submitted to the Faculty of Technology in partial fulfilment of the requirements of the award of Doctor of Philosophy.

January 2018
Abstract

We address the problem of spatiotemporal human action detection in videos. Our proposed model is able to classify and localise (in space and time) multiple co-occurring action instances in temporally untrimmed videos. Action detection in video has several practical real world applications such as human-robot interaction, autonomous driving, autonomous robotic assistant surgeon, VR (virtual reality) gaming to mention a few. In this context action detection reduces to solving three different sub-problems: 1) action classification, 2) spatial detection and 3) temporal detection. Given an input video the detection model is supposed to classify each action instance, localise them spatially using bounding boxes and detect the temporal extent of each action instance by inferring the start and end times.

Inspired by the overwhelming success of R-CNN (regions with CNN features) object detectors [1], previous approaches [2,3] adopted the R-CNN framework for action detection. Although the use of R-CNNs triggered a significant performance improvement, the downsides of relying on them are that: 1) this is computationally expensive, 2) it requires multistage training and 3) relies on unsupervised region proposal algorithms such as Selective Search [2] or EdgeBoxes [3].

Unlike [4,5], R-CNNs do not share computation and perform a separate forward pass for each object proposal, which is expensive in terms of both time and computing resources. For example, during training this requires to execute 2000 conv forwards passes for 2000 Selective Search proposals, amounting to relatively longer training time and more GPU resources than [4,5]. Besides, the R-CNN framework follows a multistage training pipeline which includes (a) fine-tuning a CNN, (b) extracting CNN features, (c) caching features to disk, and (d) training a number of one-vs-all SVMs, resulting in a very computationally expensive pipeline. Furthermore, the detection accuracy of R-CNNs is limited by their relying on unsupervised region proposal algorithms [2,3] which, besides being resource-demanding, cannot be trained for a specific detection task and are disconnected from the overall classification objective.

To overcome these issues we propose a novel action detection pipeline [6] which adopts, instead, the Faster R-CNN [7] network. Faster R-CNN is computationally less expensive than R-CNN as it shares computation during the forward pass, avoid multistage training and thus eliminating the need for feature extraction, caching and SVM training. In addition, it uses a Region Proposal
Network (RPN) to generate region proposals. Unlike unsupervised region proposal algorithms [2,3], RPN is fully-supervised and can be trained on a specific action detection dataset, leading to better quality region proposals with higher recall-to-IoU [7].

Our action detection pipeline [6] is composed of three stages. In stage 1, appearance and motion detection networks are employed to localise and score actions from RGB and optical flow maps. In stage 2, the appearance network detections are boosted by combining them with the motion detection scores, in proportion to their respective spatial overlap. In stage 3, sequences of detection boxes most likely to be associated with a single action instance, called action tubes [2], are constructed by solving two energy optimisation problems via dynamic programming. While in the first pass, action paths spanning the whole video are built by linking detection boxes over time using their class-specific scores and their spatial overlap, in the second pass, temporal trimming is performed by ensuring label consistency for all constituting detection boxes. Unlike [2], the proposed action detection pipeline performs both spatial and temporal action detection. Experimental results show that it is at least 2× faster in training and 5× faster in test time detection speeds as compared to [2,3]. Moreover, our pipeline consistently outperforms previous state-of-the-art results on the UCF-101 [8] and J-HMDB-21 [9] datasets.

Although our proposed model [6] gives state-of-the-art action detection results, it still provides a sub-optimal solutions to the problem, as it relies on seeking frame-level detections, to later compose them into ‘action tubes’ in a post-processing step. In our second model, we take a step further towards the design and implementation of a deep network architecture [10] able to classify and regress whole video subsets, so providing a truly optimal solution of the action detection problem. We propose a novel deep net framework able to regress and classify 3D region proposals spanning two successive video frames, whose core is an evolution of classical region proposal networks [7]. As such, our 3D-RPN net is able to effectively encode the temporal aspect of actions by purely exploiting appearance, as opposed to methods which heavily rely on expensive flow maps. The proposed model is end-to-end trainable and can be jointly optimised for action localisation and classification in a single step. At test time the network predicts ‘micro-tubes’ encompassing two successive frames, which are linked up into complete action tubes via a new algorithm which exploits the temporal encoding learned by the
network and cuts computation time by 50%. Promising results on the J-HMDB-21 and UCF-101 action detection datasets show that our model does outperform the state-of-the-art when relying purely on appearance.

Another promising approach to detect human actions is to perform both detection and instance segmentation in which multiple concurrent actions of the same class may be segmented out of an image sequence. The current state-of-the-art methods \cite{3, 6, 10, 11} can only perform detection and they lack instance segmentation capability. **By taking advantage of human motion segmentation work \cite{12}, we are able to associate each action tube with class-specific segmentations.** We demonstrate the performance of our algorithm \cite{13} on the challenging LIRIS-HARL action detection dataset \cite{14} and achieve a new state-of-the-art result which is 14.3 times better than previous methods. We generate action frame proposals based on the power set of connected components in the foreground-background segmentation. This means that we can output a pixel-level action instance segmentation in addition to detection with tubes. To the best of our knowledge our algorithm provides the best human action detection results on the most challenging dataset available to date. Lastly, we are the first to show qualitative action instance segmentation results.
Contents

Abstract i

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Motivating Real world applications of action detection 2
1.2 What is a spatiotemporal action instance? 3
1.3 Challenges in spatiotemporal human action detection 3
1.4 Dissertation outline 6
1.5 Avenues of investigation 8

1.5.1 Deep feature representation of spatial action instances 9
1.5.2 Deep feature representation of spatiotemporal action instances 12
1.5.3 Deep learning of action micro-tubes using 3D proposal regression 15
1.5.4 Inter-frame data association and temporal detection 16
1.5.5 Spatiotemporal action instance segmentation 18
1.5.6 Two-stream hypothesis - fusion of appearance and motion cues 20
1.5.7 Capturing inter-frame motion pattern using optical flow 20
1.5.8 Capturing motion pattern in video subset using stacked optical flow maps 21

1.6 Contribution 21

1.7 Resulting publications, softwares and media 24

1.7.1 List of Publications 24
1.7.2 List of Softwares 24
1.7.3 In the media 24
2 Related work

2.1 Action classification ................................................. 26
  2.1.1 Shallow representation ........................................ 27
  2.1.2 Human location centric approach .............................. 28
  2.1.3 Deep representation ........................................... 29

2.2 Action detection ..................................................... 30
  2.2.1 Temporal action detection ..................................... 30
  2.2.2 Spatio-temporal action detection .............................. 32

2.3 Action instance segmentation ....................................... 36

3 Datasets and evaluation metric .................................... 38
  3.0.1 Performance indicators ......................................... 39

4 Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos ................................................. 41

4.1 Introduction ......................................................... 41

4.2 Overview of the approach .......................................... 42

4.3 Methodology ......................................................... 44
  4.3.1 Region Proposal Network ....................................... 44
  4.3.2 Detection network .............................................. 45
  4.3.3 Fusion of appearance and motion cues ......................... 46
  4.3.4 Action tube generation ......................................... 46

4.4 Experimental validation and discussion .......................... 48
  4.4.1 Evaluation metrics ............................................... 48
  4.4.2 Performance comparison on UCF-101 .......................... 49
  4.4.3 Performance comparison on J-HMDB-21 ....................... 50
  4.4.4 Performance comparison on LIRIS-HARL ...................... 50
  4.4.5 Comparative analysis of region proposal quality ............. 52
  4.4.6 Ablation study ................................................. 52
  4.4.7 Impact of label smoothing on detection performance ........ 54
  4.4.8 Additional qualitative detection results ..................... 55
  4.4.9 Computing time analysis for training and testing .......... 55
  4.4.10 Discussion ...................................................... 58

4.5 Implementation details ............................................ 58
  4.5.1 Generating correct ground-truth annotations .................. 58
  4.5.2 Modifications in the existing codebase ....................... 58
  4.5.3 Optical flow based video frame generation .................... 59
  4.5.4 Selective Search region proposals ............................ 59
5 Spatio-temporal Action Instance Segmentation and Localisation

5.1 Introduction .................................................. 62
5.2 Overview of the approach ................................. 62
5.3 Methodology .................................................. 63
  5.3.1 Region proposal generation ......................... 63
  5.3.2 Appearance- and motion-based detection networks .. 69
  5.3.3 Training region proposal classifiers .................. 70
  5.3.4 Testing region proposal classifiers .................. 71
  5.3.5 Action tube generation and classification .......... 72
5.4 Experimental results ...................................... 72
  5.4.1 Instance classification performance - no localisation (NL) .. 72
  5.4.2 Detection and localisation performance ............... 73
  5.4.3 Performance vs detection quality curves ............. 74
  5.4.4 Additional qualitative detection results ............. 77


6.1 Introduction .................................................. 78
6.2 Overview of the approach ................................... 80
6.3 Network Architecture ....................................... 81
  6.3.1 Convolutional Neural Network ....................... 82
  6.3.2 3D region proposal network ......................... 82
  6.3.3 3D region proposal sampling ....................... 83
  6.3.4 Bilinear Interpolation ............................... 84
  6.3.5 Fully connected layers ............................... 85
6.4 Network training ............................................ 85
  6.4.1 Multi-task loss function ............................. 85
  6.4.2 Optimisation .......................................... 87
6.5 Action-tube generation ................................... 87
6.6 Experimental setting ...................................... 88
  6.6.1 Evaluation metrics ................................. 88
  6.6.2 Training data sampling strategy .................... 89
6.7 Model evaluation .......................................... 89
  6.7.1 Impact of different positive IoU thresholds on detection performance .......... 89
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.2</td>
<td>Impact of our training data sampling strategy on detection performance</td>
<td>90</td>
</tr>
<tr>
<td>6.7.3</td>
<td>Impact of exploiting appearance features</td>
<td>90</td>
</tr>
<tr>
<td>6.7.4</td>
<td>Detection performance comparison with the state-of-the-art</td>
<td>93</td>
</tr>
<tr>
<td>6.8</td>
<td>Supporting experiments and discussion</td>
<td>94</td>
</tr>
<tr>
<td>6.8.1</td>
<td>Impact of different fusion methods</td>
<td>94</td>
</tr>
<tr>
<td>6.8.2</td>
<td>Impact of the number of predicted 3D proposals</td>
<td>94</td>
</tr>
<tr>
<td>6.8.3</td>
<td>Loss function hyper-parameters</td>
<td>95</td>
</tr>
<tr>
<td>6.8.4</td>
<td>Ablation study</td>
<td>95</td>
</tr>
<tr>
<td>6.8.5</td>
<td>Computing time required for training/testing</td>
<td>96</td>
</tr>
<tr>
<td>6.8.6</td>
<td>Discussion</td>
<td>97</td>
</tr>
<tr>
<td>6.8.7</td>
<td>Qualitative results</td>
<td>97</td>
</tr>
<tr>
<td>6.9</td>
<td>Implementation details</td>
<td>98</td>
</tr>
<tr>
<td>6.9.1</td>
<td>Mini-batch sampling</td>
<td>99</td>
</tr>
<tr>
<td>6.9.2</td>
<td>Data preprocessing</td>
<td>99</td>
</tr>
<tr>
<td>6.9.3</td>
<td>Data augmentation</td>
<td>99</td>
</tr>
<tr>
<td>6.9.4</td>
<td>Training batch</td>
<td>100</td>
</tr>
<tr>
<td>6.9.5</td>
<td>Training iteration</td>
<td>100</td>
</tr>
<tr>
<td>6.10</td>
<td>Fusion methods</td>
<td>100</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>105</td>
</tr>
<tr>
<td>7.2</td>
<td>Network architecture</td>
<td>106</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Appearance streams</td>
<td>106</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Motion stream</td>
<td>107</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Fully connected, classification and regression layers</td>
<td>107</td>
</tr>
<tr>
<td>7.3</td>
<td>Bounding box interpolation and action tube generation</td>
<td>107</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Bounding box interpolation</td>
<td>108</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Action tube generation</td>
<td>109</td>
</tr>
<tr>
<td>7.4</td>
<td>Train and test data sampling schemes</td>
<td>110</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Training data of RGB video frames</td>
<td>110</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Training data of optical-flow maps</td>
<td>111</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Test time data sampling</td>
<td>111</td>
</tr>
<tr>
<td>7.5</td>
<td>Model Evaluation</td>
<td>111</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Impact of adding motion stream on action detection performance</td>
<td>112</td>
</tr>
</tbody>
</table>
7.5.2 Impact of bounding box interpolation on detection performance and speed ........................................... 113
7.5.3 Impact of our action tube generation algorithm on compute time ...................................................... 114
7.5.4 Comparison with the state-of-art ........................................................................................................... 114
7.5.5 Comparison of class-specific frame- and video-level APs ................................................................. 115
7.6 Implementation details ............................................................................................................................ 118
  7.6.1 Hardware and software platform ....................................................................................................... 118
  7.6.2 Data preprocessing and train data augmentation .................................................................................. 119
  7.6.3 Network weight initialisation and optimisation ................................................................................... 119
  7.6.4 Architectural modification for Motion Stream .................................................................................... 119
  7.6.5 Training iterations ............................................................................................................................... 120

8 Conclusions and future directions ............................................................................................................. 121

Appendices ................................................................................................................................................... 124

A Supporting ideas ...................................................................................................................................... 125
  A.1 Region proposal generation using 2D connected components ................................................................. 126
  A.2 Pruning Selective Search region proposals at test time ........................................................................ 127

References ..................................................................................................................................................... 128
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Spatiotemporal human action detection problem.</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Spatiotemporal human action instance segmentation.</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Spatiotemporal action instances.</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>The main challenges involved in action detection.</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>Temporal correspondence problem in action detection.</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Temporal detection problem.</td>
<td>7</td>
</tr>
<tr>
<td>1.7</td>
<td>R-CNN and Faster R-CNN architectural difference.</td>
<td>10</td>
</tr>
<tr>
<td>1.8</td>
<td>Frame-level and video-level action representation.</td>
<td>12</td>
</tr>
<tr>
<td>1.9</td>
<td>Video-level representations of actions - approach 2.</td>
<td>15</td>
</tr>
<tr>
<td>1.10</td>
<td>3D region proposals and action micro-tubes.</td>
<td>16</td>
</tr>
<tr>
<td>1.11</td>
<td>Advantages of action instance segmentation.</td>
<td>18</td>
</tr>
<tr>
<td>1.12</td>
<td>New difficulties introduced in LIRIS HARL dataset [14].</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Two-stream based deep action detection pipeline overview.</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Visualising action tube detection in a ‘biking’ video.</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Action detection/localisation results on UCF101 [6].</td>
<td>50</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample space-time action localisation results on JHMDB [6].</td>
<td>50</td>
</tr>
<tr>
<td>4.5</td>
<td>Space-time action detection results on LIRIS-HARL [6].</td>
<td>51</td>
</tr>
<tr>
<td>4.6</td>
<td>Performance comparison between Selective Search (SS) and RPN-based region proposals on four groups of action classes (vertical columns) in UCF-101. Top row: recall vs. IoU curve for SS. Bottom row: results for RPN-based region proposals.</td>
<td>53</td>
</tr>
<tr>
<td>4.7</td>
<td>Sample qualitative spatio-temporal localisation results on UCF-101.</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Action instance segmentation and detection.</td>
<td>63</td>
</tr>
<tr>
<td>5.2</td>
<td>Overview of the proposed spatio-temporal action instance segmentation and detection pipeline.</td>
<td>64</td>
</tr>
<tr>
<td>5.3</td>
<td>Visualization of the motion saliency response.</td>
<td>65</td>
</tr>
<tr>
<td>5.4</td>
<td>Supervoxel response for DPM mask and pairwise connections.</td>
<td>66</td>
</tr>
<tr>
<td>5.5</td>
<td>Visualization of the hierarchical graph based video segmentation.</td>
<td>67</td>
</tr>
</tbody>
</table>
5.6 Human motion segmentation binary maps. .............................. 68
5.7 Correct (a-c) and incorrect (d-f) instance segmentation results on
the LIRIS-HARL dataset, the correct category is shown in brack-
ets. (a) 'Try enter room unsuccessfully'. (b) 'Discussion'. (c)
‘Unlock enter/leave room’. (d) ‘Handshaking’ (Give take object
from person). (e) ‘Discussion’ (Leave bag unattended). (f) ‘Put
take object into/from desk’ (Telephone conversation). ............ 73
5.8 LIRIS HARL performance metric - Performance vs detection qual-
ity curves. ................................................................. 75
5.9 Confusion matrix obtained by human motion segmentation(HMS)
and selective search(SS) region proposal approach. They show
the classification accuracy of HMS- and SS-based methods on
LIRIS HARL human activity dataset. HMS region proposal based
method provides better classification accuracy on the complex
LIRIS dataset. ............................................................... 76
5.10 Additional qualitative action instance segmentation and localisa-
tion results. ............................................................ 77
6.1 AMTnet: Action-Micro-Tube Regression Using Deep Architec-
ture - pipeline overview. .............................................. 79
6.2 The 3D region proposals generated by our 3D-RPN network. ... 79
6.3 3D-RPN architecture. .................................................. 81
6.4 AMTnet - micro-tube linking algorithm. ................................ 87
6.5 Spatiotemporal action detection results. Video clips (a), (b) and (c)
are test videos belong to UCF-101 action classes VolleyballSpiking,
BasketballDunk and CricketBowling respectively. ................. 102
6.6 Spatiotemporal action detection results. Video clips (a) and (b)
are test videos belong to UCF-101 action classes BasketballDunk and
CricketBowling respectively. ........................................... 103
6.7 More sample detection results on UCF-101 test videos. .......... 103
6.8 Spatiotemporal action detection results on J-HMDB-21 test videos. 104
7.1 Overview of the proposed AMTnet-Flow action detection network 106
7.2 Action micro-tube generation by linear interpolation. ............. 108
7.3 Generation of detection bounding boxes for detection micro-tubes
using linear interpolation. .............................................. 108
7.4 Train and test time data sampling schemes .......................... 110
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Quantitative action detection results (mAP) on the UCF-101 dataset.</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>Quantitative action detection results (mAP) on the J-HMDB-21 dataset.</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>Classification accuracy on the J-HMDB-21 dataset.</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Quantitative action detection results on the LIRIS-HARL dataset.</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>Quantitative action detection results on the LIRIS-HARL dataset.</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>Quantitative action detection results (mAP) on LIRIS-HARL for different $\delta$.</td>
<td>52</td>
</tr>
<tr>
<td>4.7</td>
<td>An ablation study of the spatio-temporal detection results (video APs in %) on UCF-101.</td>
<td>53</td>
</tr>
<tr>
<td>4.8</td>
<td>Spatio-temporal detection results (mAP) on UCF-101 using two different sets of $\alpha_c$ values.</td>
<td>54</td>
</tr>
<tr>
<td>4.9</td>
<td>Class specific $\alpha_c$ values for each action category in UCF-101 obtained from cross validation.</td>
<td>54</td>
</tr>
<tr>
<td>4.10</td>
<td>Training and test time detection speed comparison on UCF-101 with [2,3].</td>
<td>57</td>
</tr>
<tr>
<td>4.11</td>
<td>Test time detection speed comparison on J-HMDB-21 with [2,3].</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>Quantitative measures precision and recall.</td>
<td>74</td>
</tr>
<tr>
<td>5.2</td>
<td>Qualitative thresholds and integrated score.</td>
<td>74</td>
</tr>
<tr>
<td>6.1</td>
<td>Impact of different positive IoU thresholds on detection performance (video-mAP).</td>
<td>90</td>
</tr>
<tr>
<td>6.2</td>
<td>Impact of our training data sampling strategy on per class frame-AP at IoU threshold $\delta = 0.5$, JHMDB-21 dataset (averaged over 3 splits).</td>
<td>91</td>
</tr>
<tr>
<td>6.3</td>
<td>Impact of our training data sampling strategy on per class video-AP at IoU threshold $\delta = 0.5$, JHMDB-21 dataset (averaged over 3 splits).</td>
<td>92</td>
</tr>
</tbody>
</table>
6.4 Per class video-AP comparison at IoU threshold $\delta = 0.2$, UCF-101-24 dataset ................................................. 92
6.5 Impact of our training data sampling strategy on video-mAP, JHMDB-21 (averaged over 3 splits) ................................ 93
6.6 Spatio-temporal action detection performance (video-mAP) comparison with the state-of-the-art on J-HMDB-21 ................ 93
6.7 Spatio-temporal action detection performance (video-mAP) comparison with the state-of-the-art on UCF-101 .................. 93
6.8 Impact of element-wise mean and sum fusion methods on video-mAP for J-HMDB-21 dataset (averaged over 3 splits) .......... 94
6.9 Impact of the number of predicted 3D proposals on video-mAP for J-HMDB-21 dataset (averaged over 3 splits) .......... 95
6.10 Impact of different combinations of hyper-parameters on video-mAP for J-HMDB-21 split-1 train set ......................... 95
6.11 An ablation study on J-HMDB-21 (split-01). Video-mAP is computed at IoU threshold $\delta = 0.5$ ............................. 96
6.12 Computing time comparison for training and testing ............................ 96

7.1 Impact of adding motion stream on action detection performance, J-HMDB-21 (*) dataset .................................................. 112
7.2 Impact of adding motion stream on action detection performance, UCF-101-24 (*) dataset .................................................. 112
7.3 Impact of bounding box interpolation on action detection performance (video-mAP) .................................................. 113
7.4 Impact of bounding box interpolation on action detection speed (seconds / video) .................................................. 113
7.5 Impact of our action tube generation algorithm on compute time (in seconds per video) .................................................. 114
7.6 Comparison with the state-of-art on J-HMDB-21 dataset .................. 114
7.7 Comparison with the state-of-art on UCF-101-24 dataset .................. 115
7.8 J-HMDB-21: per class frame-AP comparison with the state-of-the-art .................................................. 116
7.9 J-HMDB-21: per class video-AP comparison with the state-of-the-art .................................................. 117
7.10 UCF-101-24: per class video-AP comparison with the state-of-the-art .................................................. 118
Chapter 1

Introduction

The spatiotemporal human action detection problem in computer vision refers to the following three sub-problems: 1) action classification, 2) spatial detection and 3) temporal detection. Given an input video, the detection model is supposed to classify each action instance, localise them spatially using bounding boxes and detect the temporal extent of each action instance by inferring its start and end time points.

Consider Figure 1.1 (top row) an input video sequence to the detection system, i.e. frames at different time steps from a “Biking” video sequence. The number on the top of each video frame denotes the frame number. Figure 1.1 (bottom row): the output of an ideal detection system, i.e. the predicted class labels, bounding boxes (spatial detection) and temporal extents (temporal detection) of three different “Biking” action instance. Each colour represents an action instance.

![Figure 1.1: Top row: input video sequence Bottom row: output of an ideal detection system.](image)

Action detection is also called action localisation in the literature.
Action instance segmentation is the problem of detecting and delineating each distinct action instance present in a video \[15\]. In other words, we assign a label to each pixel of the frame where labels are class-aware and instance-aware. Consider again the same input video sequence as shown in Figure 1.1 (top row). The output of an ideal detection system capable of performing both action detection and instance segmentation is depicted in Figure 1.2.

Figure 1.2: Output of an ideal detection system capable of performing action instance segmentation.

### 1.1 Motivating Real world applications of action detection

Spatiotemporal human action detection in video is one of the most demanding research areas in computer vision as it has several practical real-world applications. In autonomous driving \[16,17\], the space-time localisation of human actions will allow the system to detect different categories of actions, e.g. “Walking” and “Running”, and thus enables the vehicle controller to adjust speed and course accordingly. In robot-assisted surgery \[18,19\], an accurate action detector enables the robot to provide the surgeon an environment to carry out complex procedures with better precision and control. Furthermore, the quality of human-robot interaction and virtual reality gaming can be improved further with smarter action detection algorithms. The detection algorithms can be categorised as either “offline” or “online”. Those detection systems which require the entire video clip beforehand to generate detection results fall into the offline category, and those which are able to generate detections from a small subset of the video and can incrementally link those detections over time as more and more frames are visible to the system are categorised as online. For example, processing streaming video frames online.

The action detection pipelines presented in Chapter 4 and Chapter 5 are suitable for offline applications as they follow multi-stage detection process and are relatively expensive. whereas, Chapter 6 and Chapter 7 are devoted to design
faster and single-stage end-to-end trainable action detection frameworks which can be easily extended to online versions just by replacing their offline action tube generation algorithms with the online tube building algorithm \[20\].

1.2 What is a spatiotemporal action instance?

Before answering the above question, we might be interested to know even: “what is an action?”. A very nice explanation of this question can be found in Section 1.2 of \[21\].

In a given input video, an action may present in any spatial and temporal locations. Besides, multiple co-occurring instances of the same or different action class can be present in a video. Consider Figure 1.3 (a) and (b) in which multiple action instances of the same class “Fencing” (4 instances) and “Biking” (3 instances) are present in the respective video frames. In Figure 1.3 (c) and (d), multiple action instances of the two different classes “Handshaking”, “leave baggage unattended” and “Unlock enter leave room”, “put take object into from box desk” are present respectively. Note that, bounding boxes are used to spatially localise each action instance in a video frame. Each colour represents a unique action instance belongs to either same or different class. The set of bounding boxes connected over time without any whole/gap in between forms a spatiotemporal action instance as shown in Figure 1.3 (e). In action detection literature, each spatiotemporal action instance is widely known as an “action tube” \[2\]. Note that each action tube also tells us about the temporal duration/extent of that action, i.e. the start and end times.

1.3 Challenges in spatiotemporal human action detection

Spatiotemporal human action detection is a hard problem. It is extremely challenging to design a representation (for machine to understand) which is robust enough to deal with both intra-class variability and inter-class similarity problems. Other major difficulties arise from spatial and temporal localisation, and Inter-frame data association. Moreover, human actions possess high variations in geometry and topology \[22\].
Figure 1.3: Spatial localisation of action instances of class: (a) “Fencing” (4 instances), (b) “Biking” (3 instances), (c) “Handshaking” and “Leave baggage unattended”, (d) “Unlock enter leave room” and “Put take object into from box desk”. (e) spatiotemporal localisation of two “Fencing” action instances, where \( t_1 < t_2 < t_3 < t_4 \). Dotted lines represent temporal association between action instances over time.

**Intra-class variability.** The appearance and motion of a particular class may differ significantly due to variation in illumination, camera motion and viewing angle, partial occlusions, background, scale. In addition, action instances belong to same class may differ due to a large variability in the execution style of an action such as poses, motion dynamics, contextual information, speed etc. For instance, the two action instances in Figure 1.4 (a) and (b) have variation in pose, camera viewing angle, contextual information (a door and a bench) yet they belong to the same action class “Sit”.

**Inter-class confusion.** There are cases, where instances from different action categories possess similar: (a) appearances, (b) contextual information (e.g. players in both “Basketball” and “Basketball dunk” share similar appearances and in both actions basketballs may present as the contextual information), (c) motions and (d) poses (e.g. both actions “Stand” and “Sit” may shares similar motions and poses). Some other interesting examples are: run and walk, laugh and yawn, crawl and swim where it is hard to have a discriminative representation.

Thus, learning representation which has good generalisation ability over wide range of actions belong to same class, and yet enough discriminative between actions of different classes is challenging [23]. Figure 1.4 highlights some of the above problems (video frames are taken from action detection datasets: UCF-
Chapter 1. Introduction

Figure 1.4: (a-b): intra-class variability - the two action instances have variation in pose, camera viewing angle, contextual information (a door and a bench) yet they belong to the same action class “Sit”. (c-d): in (c), the “Biking” instance depicted with a red bounding box is hard to detect due to large variation in scale; in (d), the “Biking” instance localised by a green coloured bounding box is partially occluded by the blue coloured one and thus difficult to detect. (e-f): inter-class confusion - though these two action instances share similar poses, motions, contextual information (chairs) they belong to two different classes “Stand” and “Sit”. (g-h): these action instances are hard to detect due to poor illumination (g) and motion blur (h) respectively.

Spatial localisation. Finding the spatial locations of action instances present in a video is challenging due to several factors: (a) the spatial locations of action instances may vary over time, (e.g. the locations of the “Biking” instance \(d_3\) and the “Fencing” instance \(d_1\) change over time as shown in Figure 1.1 and Figure 1.3 respectively; (b) the actor is partially occluded by other actor or object (e.g. the one of the Bikers (located with green bounding box) is partially visible due to occlusion as shown in Figure 1.4 (d)); (c) presence of co-occurring action instances belong to same or different action classes (Figure 1.3); (d) in addition, an actor can perform multiple actions in parallel (e.g. a person is “Waving his hand” while “Walking”).

Inter-frame data association. Consider Figure 1.5. It shows a video with a sequence length of 80 frames \((f_1 \text{ to } f_{80})\) in which there are two action instances. To correctly detect these two instances, one approach would be to associate or link the frame-level detections \((d_1 \text{ and } d_2)\) in time to obtain two valid action tubes. The temporal linking of detections in action detection can be considered as the data association problem \([24]\) problem in “multi-target tracking” domain \([3]\). However, in tracking data association is done for detections belong
to single class such as pedestrian, whereas in action detection data association is
to be performed for detections belongs to multiple human action categories. In
Figure 1.5, the correct inter-frame data associations for each action instance is
depicted using blue and red solid lines. Data association is hard because of the
following main reasons. [Note for Fabio: please give a pass.] (a) As the number
of frame-level detections \( m \) and the sequence length \( n \) of an input video increase
the running time complexity of the data association algorithm also increases e.g.
in case of a Viterbi algorithm the complexity increases in polynomial time, the
running time is \( O(nm^2) \) \[25\]. (b) Another challenge is to make the data as-
sociation algorithm robust to position-swapping (Figure 1.5) and associate the
inter-frame detections in time correctly (e.g. the locations of action instances \( d_1 \)
and \( d_2 \) are interchanged between frame \( f_{20} \) to \( f_{40} \)). (c) lastly, the data associ-
ation problem becomes more harder due to partial occlusion, sudden-change in
illumination, camera motion and/or viewing angle.

**Temporal localisation.** Modelling a system which can correctly infer the
temporal extent (start and end time points) (Figure 1.5) of each action instance
present in a video is extremely difficult: (a) action instances may appear and
disappear within a video at any time point. Consider Figure 1.1. The “Bik-
ing” action instances \( d_1 \) and \( d_2 \) disappear after frames 161 and 176 respectively.
Whereas, the \( d_3 \) starts at frame 161. Even though the actor is present through-
out the entire video, the actual start and end time of the action may be at any
time point within the duration of the video (Figure 1.6) which make the tempo-
dral detection problem more harder. [Note for Fabio: please give a pass.] A naive
sliding window algorithm to detect the temporal extent of actions will have a \( n! \)
run time complexity where \( n \) is the number of frames in a video.

In spite of these aforesaid challenges in action detection, substantial research
initiatives have been taken in this area \[2,3,26–30\]. For a detailed list of works,
please refer Chapter 2. In the following section, we will present the outline of
this thesis.

### 1.4 Dissertation outline

[Note for Fabio: please give a pass to this section.]  
First and foremost, we introduce the human action detection datasets used
throughout this work in Chapter 3. Next, we formulate the spatiotemporal
action detection in Chapter 4 where frame-level RGB and optical flow image data
Figure 1.5: Linking frame level bounding boxes to build two action tubes. Each action tube is assigned a unique id, here ids are 1 and 2. Two different colours (red and blue) are used to denote two different action tubes. Solid lines show valid detections and temporal links whereas, the dotted lines denote detections and links discarded during temporal localisation of action instances. Note that, in frame 40 \( f_{40} \) the tube ids get swapped which is hard to track. Also note that the start and end times are different for two action tubes, and detecting these temporal extent/duration of each tube is again challenging.

Figure 1.6: Temporal detection problem: Even though the actor is present throughout the entire video the actual “Basketball” action starts at \( t_4 \) and ends at \( t_8 \) which makes the temporal detection more difficult.

are mapped to a feature space using two-stream convolutional neural networks (CNNs) architecture \cite{31} and subsequently, these CNN features are used for action classification and bounding box regression. In addition to predicting the frame-level confidence score and spatial location of each action instance in a video, the proposed framework also links the detections in time and apply temporal label smoothing to solve for temporal localisation.

In Chapter 5 we consider the problem of spatiotemporal action instance segmentation, where the task is to detect and delineate each action instance by assigning each pixel a label. Unlike the bounding-box level class-aware action labels in Chapter 4, the labels obtained from instance segmentation are at pixel-level and they are both class- and instance-aware.
We present a more optimal solution to the problem of action detection in Chapter 6. Note that the action detection framework presented in Chapter 4 can only provide a sub-optimal solution to the problem as it relies on training CNNs which can predict only frame-level class confidence scores and bounding boxes. As these CNNs trained on individual video frames, they do not learn the temporal associations between inter-frame action instances. The new action detection framework presented in Chapter 6 addresses this limitation by training CNNs on a subset of video frames and predicting video-level confidence scores and detections (i.e. action “micro-tubes” (§1.5.3)).

An extension of the work in Chapter 6 to incorporate motion features, and predict action micro-tubes over longer distanced pairs of videos frames is presented in Chapter 7. Moreover, the detection frameworks presented in Chapter 6 and Chapter 7 are computationally efficient, faster and adapts single-stage end-to-end training and testing strategies. In contrast, detection pipelines presented in Chapter 4 and Chapter 5 are relatively expensive and slower, follow multi-stage training and testing strategies, thus not suitable for online applications.

1.5 Avenues of investigation

“Features matter”! Powerful visual features or image representation is the key to the success of any computer vision algorithm such as image classification and object recognition systems [32]. Over the past several years, the improvements in the performance of computer vision based systems can be attributed to the evolution of effective data representation starting from early Bag-of-Visual-Words (BoVW) [33,34], to Improved Fisher Vector (IFV) [35], and more recently, the deep feature representation of Convolutional Neural Networks (CNNs) [36]. In recent years, the deep feature representation has demonstrated significant quantitative improvements in image understanding over classical representation (e.g. BoVW, IFV etc.) [32].

Unlike still images, videos contain highly dynamic appearance and motion patterns, thus, problems such as human action detection in videos require effective visual representation which is robust to the time varying visual data. Further, as discussed earlier (cf. Section 1.3), the two main challenges in action detection are: (1) inter-frame data association and (2) temporal action localisation. Majority of the action detection approaches try to solve these problems using a graph-based video segmentation and sliding window technique (cf. Chap-
Chapter 1. Introduction

In this work, we mainly focus on exploiting deep features for image and video data representation and posing the inter-frame data association and temporal localisation as energy optimisation problems which can be efficiently solved using Viterbi algorithm. In the following subsections, we introduce frame-level (§1.5.1) and video-level (§1.5.2, 1.5.3) deep feature representation used in this work, combined with the original methods proposed (§1.5.4, 1.5.5) to address the spatiotemporal action detection problems set out in Section 1.4.

1.5.1 Deep feature representation of spatial action instances

Unlike action recognition, our goal is to both localise (in space and time) and classify action instances in temporally untrimmed videos. Two promising work in this direction were proposed by Gkioxari and Malik [2] and Weinzaepfel et al. [3]. They used CNN based deep feature representation to encode both spatial and temporal features of human actions to detect action tubes. The success of these work [2,3] are mainly due to: (1) use of regions with CNN (R-CNN) features [1] for video data representation which not only provide better feature encoding of human actions but also improves the spatial localisation accuracy with the help of region proposals [37]; (2) use of two-stream CNN approach [31] to effectively capture the complementary information from video data i.e. frame-level appearance and inter-frame motion dynamics associated with human actions.

However, the drawbacks of these approaches [2,3] arise due to the fact that: 1) they are computationally expensive, 2) require multi-stage training and 3) rely on unsupervised region proposal algorithms such as Selective Search [2] or EdgeBoxes [3]. The main reason of these aforesaid drawbacks is that, these action detectors [2,3] rely on R-CNN object detection framework. Unlike [4,5], R-CNNs do not share computation and perform a separate forward pass through convolutional layers for each object proposal (see Figure 1.7 (a)), which is expensive in terms of both time and computing resources. For example, during training R-CNNs require to execute 2000 conv forwards passes for 2000 Selective Search proposals, amounting to relatively longer training time and more GPU resources than [4,5]. Besides, the R-CNN framework follows a multi-stage training pipeline which includes (a) fine-tuning a CNN, (b) extracting CNN features, (c) caching features to disk, (d) training a number of one-vs-all SVMs and fi-
Figure 1.7: R-CNN and Fast R-CNN architectural difference.

nally (e) solving a regression problem to fit the region proposal bounding-boxes as per the ground-truths, resulting in a very computationally expensive pipeline. Furthermore, the detection accuracy of R-CNNs is limited by their relying on unsupervised region proposal algorithms [2, 3] which, besides being resource-demanding, cannot be trained for a specific detection task and are disconnected from the overall classification objective.

For instance, On large datasets such as UCF-101 [8], Gkioxari and Malik [2]’s action detection pipeline takes a week for training and feature extraction [2] plus one extra day for SVM training. At test time, detection is slow as features need to be extracted for each region proposal via a CNN forward pass. Moreover, Gkioxari and Malik [2]’s work does not address temporal localisation, whereas Weinzaepfel et al. [3]’s sliding window approach for temporal action detection is relatively expensive.

To deal with these limitations, we explored the possibility of modelling a deep feature based action detection framework which is capable of sharing computation during a forward pass (see Figure 1.7 (b)), can avoid multi-stage training by training a CNN for both action classification and bounding box regression. Such a detection system is bound to be computationally less expensive than that of [2, 3] and eliminates the need for feature extraction, caching and SVM training. Moreover, current spatiotemporal human action detection methods [2, 3] use un-

\[\text{For feature extraction, we used 7 Nvidia Titan X GPUs in parallel.}\]
supervised region proposal generation algorithms (such as Selective Search \cite{2011:48} and EdgeBoxes \cite{2015:38}) to generate rectangular region hypotheses. As these are unsupervised algorithms, they can not utilise the ground-truth action location information provided with a specific dataset. By leveraging on deep CNN features and posing the region proposal generation as a supervised machine learning problem, we can generate better quality action region hypotheses with higher recall-to-IoU \cite{2016:7}. Besides, to generate region proposals using these unsupervised methods, each video frame is to be processed individually which is indeed computationally expensive. Whereas, in a supervised approach \cite{2016:7}, this expensive per-frame region proposal generation process can be completely eliminated using a fixed set of anchor boxes. Therefore, in order to improve the detection accuracy and reduce the computing cost, it is desirable to model an action detection framework which can (a) generate high quality region proposals by taking advantage of deep representation under a supervised setting and (b) jointly optimise both the region proposal and action detection objectives.

This idea drives us towards designing a novel deep network based action detection framework (cf. Chapter \ref{chapter:4}) which can generate region proposals by optimising jointly a binary “actionness” \cite{2014:39} classification and a bounding-box regression objectives by exploiting deep features combined with the ground-truth class labels and spatial location information associated with each action instance present in the training videos. The “actionness” classification objective assigns high scores to those proposals which are highly likely to contain an action instance (positive training samples) and low scores otherwise, and the box regression objective helps to improve the localisation accuracy of the positive training samples by regressing them towards the corresponding ground-truth boxes. Once the region proposals are obtained, they can be sorted as per their actionness scores and the top $k$ proposals then can be used to train a deep network for action classification and bounding-box regression.

We refer the visual representation mentioned above as “frame-level” representation because it encodes only the static appearance of the actor(s) and the scene from a single video frame, but it fails to encode the motion pattern inherently associated with the action.

**An illustration of a frame-level deep action representation.** Consider Figure \ref{fig:1.8} (a). A cropped image patch is passed as input to a CNN which computes deep convolutional features encoding the static appearance of the path. Subsequently, the high-level feature encoding is used to train a classifier for action recognition. Note that, this feature encoding encodes only the static appearance
cues due to the fact that it belongs to a single video frame. In the following subsection, we discuss the “video-level” representation to encode both spatial and temporal features of an action instance within a video.

### 1.5.2 Deep feature representation of spatiotemporal action instances

The most recent deep learning based action detection frameworks \cite{2, 3, 6, 11} exploit the frame-level action representation (Section 1.5.1), along with region proposal algorithms \cite{7, 37, 38} and two-stream architecture \cite{2, 31}. These methods first construct training hypotheses by generating region proposals (or ‘regions of interest’, ROI\(^3\)), using either Selective Search \cite{37}, EdgeBoxes \cite{38} or a region proposal network (RPN) \cite{7}. ROIs are then sampled as positive and negative training examples as per the ground-truth. Subsequently, CNN features are extracted from each region proposal. Finally, ROI pooled features are fed to a softmax and a regression layer for action classification and bounding box regression, respectively.

This dominant paradigm for action detection \cite{2, 3, 6, 11}, however, only provides a sub-optimal solution to the problem. Indeed, rather than solving for

\[
T^* = \arg \max_{T \subseteq V} \text{score}(T), \tag{1.1}
\]

where \(T\) is a subset of the input video of duration \(D\) associated with an instance

\(^3\)A ROI is a rectangular bounding box parameterized as 4 coordinates in a 2D plane \([x_1, y_1, x_2, y_2]\).
of a known action class, they seek partial solutions for each video frame

\[ R^*(t) = \arg \max_{R \subset I(t)} \text{score}(R), \]  

(1.2)

to later compose in a post-processing step partial frame-level solutions into a solution

\[ \hat{T} = [R^*(1), \ldots, R^*(D)] \]  

(1.3)

of the original problem (1.1), typically called action tubes [2]. By definition,

\[ \text{score}(\hat{T}) \leq \text{score}(T^*) \]  

(1.4)

and such methods are bound to provide suboptimal solutions. More specifically, in the post-processing step frame-level detection bounding boxes are linked in time to build action tubes either by using a Viterbi [2,6,11] or a tracking [3] algorithm. This post-processing step is essential as those CNNs do not learn the temporal associations between region proposals belonging to successive video frames (i.e. the inter-frame data associations (Section 1.3)). For instance, an “archery” action can be easily identified only from a still video frame due to its salient appearance cues like the presence of bow and arrow. However, actions with similar appearance features such as: “crawl”, “breaststroke” and “swim”, “laugh” and “yawn”, “walk” and “run” are hard to discriminate only using still frames as they might be sometimes ambiguous due to their inter-class confusion (cf. Section 1.3), and thus, there is a need to incorporate motion features to achieve more discriminative action representation. To compensate for this and learn the temporal dynamics of human actions, the two-stream architecture makes use of optical flow based CNN feature representation (Section 1.5.6) to encode the motion pattern of human actions. Although, the flow based CNN feature representation helps to improve the action detection accuracy, the architecture of these CNNs have limited temporal scale as the networks operate on either only a pair of consecutive optical flow frames [2,3,6] or a stack of 5 flow maps [11]. This frame-level representation is mostly suitable for object detection, but inadequate for action detection where both spatial and temporal localisation are crucial.

These aforementioned limitations motivate us to design a new deep learning architecture (cf. Chapter 6) where (a) representations of space-time action instances are learnt from subsets of video frames (i.e. “video-level” representations), (b) space-time 3D region proposals (§ 1.5.3) are exploited for video-level
training as opposed to 2D region proposals used for frame-level training and (c) action micro-tubes (§1.5.3) are temporally linked as opposed to linking of frame-level detection bounding-boxes (§1.5.4).

Another drawback with these two-stream based methods [2,3,6,11] is that the network can not learn the pixel-wise correspondences between appearance and motion features as the fusion is performed only at test time [40]. We extend the deep architecture presented in Chapter 6 by leveraging on the video-level action representation combined with a train time fusion scheme to fuse appearance and motion cues which allows the network to learn the pixel-wise correspondences between both RGB and flow features (cf. Chapter 7).

Illustrations of video-level action representation:

**Video-level representation - approach 1.** Now, consider Figure 1.8 (b) where a video-level action representation (cf. Chapter 7) is illustrated. Two cropped image patches belong to two successive video frames ($f_t$ and $f_{t+\Delta}$) are processed through two separate CNNs (weights are not shared among these two CNNs) The output feature maps of these two CNNs are fused using an element-wise sum fusion [40], and subsequently the fused feature representation is used to train for action classification. Again note here, this feature representation encodes both the appearance and motion cues as the encoding belongs to a pair of successive video frames.

**Video-level representation - approach 2.** Another variant of video-level action representation [41] is shown in Figure 1.9 Cropped image patches belong to a set of video frames ($f_1, f_2\ldots f_{t+\Delta}$) are processed through their corresponding CNNs (weights are shared among these CNNs) and subsequently the output feature maps are concatenated to obtain a single feature representation.

The main drawback of approach-2 is that the dimensionality of the concatenated feature representation increases linearly with the number of video frames leading to a very high dimensional feature representation in case of longer video subsets. Further, as the weights are shared among CNNs, they might not learn the salient temporal cues independently. Whereas, in approach-1, as the weights are not shared between those two CNNs, we expect them to learn independently the different motion cues at two different time points $t$ and $(t + \Delta)$. Hence, in this work, we use approach 1 for video-level representation.
1.5.3 Deep learning of action micro-tubes using 3D proposal regression

Frame-level region proposals \([2, 3, 6, 11]\) are rectangular region hypotheses used to train a bounding-box regressor for spatial action localisation. Unsupervised region proposal algorithms \([37, 38]\) were heavily exploited \([2, 3]\) to generate action region hypotheses.

Due to the fact that this proposal generation algorithms are unsupervised and are disjoint from the overall training optimisation, a region proposal network (RPN) was proposed \([7]\). RPN is a fully convolutional neural network which generates region proposals by first initialising the 2D image search space with some predefined anchor boxes, and subsequently, regressing the best matched (positive) anchor boxes towards the ground-truth based on the location-specific convolutional features. The upside of RPN are: (a) it is fully supervised and (b) can be integrated into the optimisation process, leading to relatively higher recall-to-IoU \([7]\). RPN is thus suitable for generating 2D region proposals for frame-level representation and training.

For video-level action representation (cf. Section 1.5.2) and training, we require action region hypotheses which span in both space and time. To this end, we propose a 3D-RPN network (cf. Section 6.3.2) to generates action region hypotheses spanning both space and time by extending the RPN network \([7]\). Below we explain the concept of a 3D region proposal and an action micro-tube with an example which helps the readers to understand the action detection frameworks presented in Chapter 6 and Chapter 7.

Consider Figure 1.10 (a). A frame \(f_t\) from a “Diving” video sequence is shown (taken from UCF-101-24 action detection dataset \([8]\)). The ground-truth
bounding-box is shown in green and a closely matched anchor box (or a 2D region proposal) is shown in red. In frame-level action representation and training, a regression loss is minimised to regress the red anchor box towards the green ground-truth. Now consider Figure 1.10 (b) where a “Diving” action spans over two successive video frames $f_t$ and $f_{t+\Delta}$. In this case, a ground-truth action **micro-tube** (i.e. a pair of ground-truth boxes belongs to frames $f_t$ and $f_{t+\Delta}$ respectively) is shown in green. One of the best matched **3D region proposals** (i.e. a pair of anchor boxes which has high mean overlap with the corresponding ground-truth boxes) is shown in red. During video-level training, the red 3D region proposal is regressed towards the green ground-truth micro-tube.

![Figure 1.10: Space-time action localisation. (a): regressing an anchor box (in red) towards the ground-truth bounding-box (in green); (b): regressing a 3D region proposal (in red) towards the ground-truth micro-tube (in green); (c): linking detection micro-tubes in time at test time.](image)

Lastly, unlike frame-level action detection methods [2, 3, 6, 11], which outputs detection bounding-boxes at test-time, an action detection model trained on video-level features outputs detection micro-tubes (cf. Chapter 6). Temporal linking of detection micro-tubes are faster than linking of frame-level detections. Figure 1.10 (c) shows the temporal linking of the action micro-tubes extracted during test time.

### 1.5.4 Inter-frame data association and temporal detection

In action detection, the inter-frame temporal association or temporal correspondence (of action instances) problem is mostly solved by using graph-based video segmentation methods and shallow features (e.g. dense trajectories [42]) [26, 27, 29, 43]. These approaches rely on highly expensive and time consuming steps: extraction of supervoxels (at diff. levels of segmentation hierarchy) and dense trajectory features. More recently, the temporal linking of frame-level de-
tects (or the data association problem) (Section 1.3) is performed either by solving an optimisation problem using Viterbi algorithm as in [2, 6] or applying tracking-by-detection approach [3]. One highly successful approach is to pose the temporal linking as an optimisation problem [2] in which frame-level detections are linked in time as per the overlaps of their spatial locations and their class specific confidence scores. Another promising solution is the tracking-by-detection approach where, the best frame-level detection is tracked over the entire video. The key factors affecting the performance of a tracking-by-detection approach are: (1) a robust selection criteria for picking the best detections to track and (2) an effective initialisation technique for the tracker [3].

The results obtained after solving the above temporal linking problem is a set of class-specific action tubes (or tracks) which do not explicitly carry any information about the start (initiation) and end (termination) time points of their respective action instance. To detect the start and end times of each human action instance present in a video, we need to design a model which can perform temporal detection (or localisation) for us [1, 3].

A common approach for temporal detection is to apply sliding window on action tubes (tracks) [3] at different temporal scales. For instance, Weinzaepfel et al. [3] use 13 different temporal scales (e.g. window length of 20, 30, ..., 600).

A sliding window is expensive due to a large search space along different temporal scales. The runtime complexity of a sliding window algorithm is $O(T^2)$ where $T$ is the duration of the video. For longer video sequences, traversing through this huge search space is computationally expensive and not suitable for applications require online and real-time performance.

A simple and more robust solution is to pose the temporal detection as an energy optimisation problem [6] and apply temporal label smoothing (as in Evangelidis et al. [44]) for each action tube individually. By posing it as a label smoothing problem, we reduce the algorithm’s runtime complexity from polynomial time $O(T^2)$ (for sliding window) to linear time $O(T)$. Thus, can design more cost effective algorithm also suitable for online realtime applications.

Evangelidis et al. [44] solve for a multi-label smoothing problem, i.e. applying label smoothing to detect the temporal extents of 25 different gestures in videos. The label smoothing is performed based on the 25 frame-level confidence scores. In our case, as we have action tubes, each of which belongs to a particular action category $c$, we can pose the temporal action detection as a binary label smoothing problem. We can thus obtain a temporally trimmed action tube by
assigning each detection box (of an action tube) either a class label \(c\) or 0 (a 0 denotes a background class) as per their class-specific confidence scores. (cf. Chapter 4).

1.5.5 Spatiotemporal action instance segmentation

[Fabio, please give a pass to this section.]

The problem of “action instance segmentation” in images can be considered as an intersection of both (a) semantic action segmentation and (b) frame-level action detection problems [45]. In semantic segmentation, each pixel in a video frame is assigned with its action class label. However, these are not instance-aware labels, i.e. different instances of the same action category can not be uniquely identified. Figure 1.11 (a) shows the output of an ideal semantic segmentation method. In this video frame, there are two instances of

![Figure 1.11: Action instance segmentation can be though of as an intersection of semantic segmentation and frame-level action detection. Output of an ideal (a) semantic segmentation, (b) frame-level action detection and (c) action instance segmentation methods. Note that, in (a), instance-aware class labels are missing. Whereas, in (c), the two different “typing on keyboard” action instances can be precisely located using pixel-level instance-aware class labels. Each colour of the overlaid masks on the video frames denotes an action class label which is not instance-aware in (a), but is indeed instance-aware in (b) and (c).](image)

“typing on keyboard” action class and one instance of “entering an office” action class. Although, semantic segmentation can successfully assign each pixel with its action class label, fails to assign instance-aware class labels to two different instances of the same class “typing on keyboard”. Frame-level action detection does provide both class- and instance-aware labels, but labeling is done at a very coarse, bounding-box level. The output of an ideal frame-level action detector is shown in Figure 1.11 (b). Action instance segmentation provides both class- and instance-aware labels at pixel-level (see Figure 1.11 (c)). Unlike seman-
tic segmentation, instance segmentation can uniquely identify instances of the same class. Unlike frame-level action detection, it assigns a label to each pixel instead of each bounding-box. Thus, action instance segmentation facilitates more accurate action recognition and localisation of human action instances which is beneficial for many real-world applications such as autonomous driving, robotics, virtual reality gaming.

Although, lot of research initiatives have been taken for action detection \[2, 3, 6, 11\], yet, we have not noticed any research work in the direction of space-time action instance segmentation. Emerging real-world applications require an all-round approach to the machine understanding of human behaviour which goes beyond the bounding-box level space-time localisation of human actions. For instance, assume a self-driving car wants to localise a particular instance of a “pedestrian” action class among many pedestrians in a crowded scene. In such scenarios, an “spae-time action instance segmentation” method can enhance the localisation capability of the self-driving car by delineating the pedestrian at pixel-level.

Moreover, the existing works \[2, 3\] perform action detection in settings where each video clip contains only a single action category \[2, 3\] (e.g. both J-HMDB-21 \[9\] and UCF-101-24 \[8\] action detection datasets have videos containing actions belong to a single class) and most of the clips are temporally trimmed by human observers. For example, Videos of 13 action categories out of 24 are temporally trimmed in UCF-101-24 dataset (cf. Section ??) [Put reference here.] and all the videos of J-HMDB-21 dataset are temporally trimmed. However, in real-world scenarios where videos often contain co-occurring actions belong to different categories and they are not temporally trimmed (i.e. the temporal extents of action instances present in a video are unknown). Consider the LIRIS-HARL human action detection dataset \[14\] which poses several challenges such as intra-class variability, inter-class confusion, (Section 1.3) since many actions (1) have similar appearance and motion whilst belonging to distinct classes such as “unsuccessfully unlocking door” and “successfully unlocking door” (see Figure 1.12 (a) and (b)), (2) have very different appearance yet share the same class such as “put/take object into/from box” (see Figure 1.12 (c) and (d)), and (3) co-occurring actions such as “discussion” and “put/take object into/from box” (see Figure 1.12 (c)).

These aforesaid drawbacks in the present detection methods motivate a detection pipeline (cf. Chapter 5) which can perform: both spatio-temporal action instance segmentation and action detection in a setting where videos are tempo-
rally untrimmed and contain multiple co-occurring action instances of different action categories.

1.5.6 Two-stream hypothesis - fusion of appearance and motion cues

Similar to [2, 31], our detection frameworks (presented in Chapters 4, 5 and 7) are also inspired by the two-streams hypothesis of human vision system. According to this hypothesis, the appearance (shape, color and texture) and motion (spatial transformations and movement) information are complementary and combining both these information leads to better understanding of the visual world. In human brain, the fusion of appearance and motion cues happens naturally. However, for machines, we need to explicitly design an algorithm to combine these two information to achieve better detection accuracy. To simulate the two-streams hypothesis for machines (computing devices), we use two convolutional neural networks (CNNs). The first CNN is used to encode the static appearance of actors and their environment from RGB video frames, we name it as appearance-based CNN. The second CNN is used to capture the pattern of motion of actors and objects (if any) associated with the action (or actions) from optical flow maps (Section 1.5.7), we name it as motion-based CNN. We further propose two effective methods for combining the appearance and motion cues: (a) a late fusion technique at test time (cf. Chapter 4) and a CNN feature fusion approach during training (cf. Chapter 7).

1.5.7 Capturing inter-frame motion pattern using optical flow

In action detection, optical flow is heavily exploited to capture the motion pattern of human actions in videos. 2, 3, 6. The motion pattern of human actions present in two consecutive video frames can be represented as a flow (or heat)
map at pixel level [2]. To generate the flow map firstly, optical flow is estimated for a pair of video frames using the widely used algorithm [47]. Secondly, magnitudes of all flow vectors are computed. Then, the optical flow signal (i.e. flow in \( x \) and \( y \) directions) and the flow magnitude are scaled up by multiplying with 16 and subsequently converted to integer values between 0 and 255. Finally, the flow map is generated by stacking the \( x \)-component, the \( y \)-component and the flow magnitude, which can be considered as a 3 channel RGB image representation of the optical flow signal.

1.5.8 Capturing motion pattern in video subset using stacked optical flow maps

coming soon ...

1.6 Contribution

Firstly, I cast the expensive multi-stage action detection approach [2,3] (cf. Section 1.5.1) into a relatively less expensive single-stage setting (cf. Chapter 4), where appearance- and motion-based CNNs are trained on RGB and optical flow data to perform frame-level action classification and spatial localisation. Subsequently, frame-level RGB and flow detections are fused and then linked in time to generate action tubes. Finally, label smoothing is applied to each action tube for temporal localisation. In order to achieve a better representation of image data and to design a single-stage detection framework, I replaced the R-CNN architecture [1] used in [2,3] with Faster R-CNN [7]. The relatively deeper representation of VGG-16 [48] (used in Faster R-CNN) is beneficial for the classification accuracy and allows our model to generalise well to a wide range of action detection datasets. I reported the outperforming action classification results (cf. Section 4.4.3) of our framework on J-HMDB-21 action detection dataset [9]. As a by-product of using Faster R-CNN, our detection network can have access to relatively better quality action proposals (predicted by a RPN network) than those generated by the Selective Search (SS) algorithm [2]. I presented a quantitative comparision (cf. Section 4.4.5) between SS- and RPN-based region proposals and demonstrated that RPN-based proposals exhibit much better recall-to-IoU than SS-based boxes. Furthermore, I proposed an original test time fusion strategy to fuse the RGB and optical flow information based on the detections’ softmax probability scores and their inter-frame spatial overlaps (cf.
Section 4.3.3. I demonstrated quantitatively that our fusion strategy significantly improves the detection accuracy (cf. Section 4.4). Besides, I reported an ablation study (cf. Section 4.4.6) which again attest the significance of our fusion method. Further, (a) to link the frame-level detections (after fusion) over time and (b) to perform temporal localisation, we bring forward a two-pass dynamic programming (2PDP) approach, where Viterbi algorithm is used to solve for these two optimisation problems (a & b). I also demonstrated the efficacy of our temporal detection algorithm (the 2nd pass of DP) which behaves gracefully and significantly improves the action detection performance (cf. Section 4.4.7). The resulting action detection system is suitable for human action classification, spatial and temporal localisation of actions in unconstrained videos.

Secondly, I proposed a variant of action detection pipeline (cf. Chapter 5) which can perform “action instance segmentation” alongside “action detection”. To allow our model to output pixel-level instance segmentation, I applied the human motion segmentation algorithm [12] on the testset videos to extract binary silhouettes of human actions in space and time. I then proposed a simple but effective region proposal algorithm (cf. Section 5.3.1) which generates region proposals based on the power set of connected components in the space-time binary silhouettes of human actions. To the best of my knowledge, this is the first work which addresses jointly both space-time action instance segmentation and action detection problems. I also demonstrated quantitatively that our proposed model outperforms the existing methods in one of the most challenging action detection datasets available to date (cf. Section 5.4). Moreover, I was the first to show qualitative action instance segmentation results.

Next as the third contribution, we proposed a new action detection paradigm (cf. Chapter 6) which on the methodological side, a key conceptual step forward from the frame-level action representation (§ 1.5.1) (heavily exploited in [2,3,6,11]) towards a video-level representation (§ 1.5.2). In place of encoding frame-level action regions to a feature space, I encoded video-level action regions (i.e. action regions span across a pair of successive frames) to a feature space using a fusion technique (cf. Section 6.3.1) which performs element-wise fusion of convolutional features computed from two successive video frames. Unlike the frame-level representation, the video-level feature encoding can effectively encodes the temporal associations between action instances present in a video subset. On the network design side, I proposed a novel end-to-end trainable deep neural network architecture which facilitates video-level feature encoding. Also this new network architecture addresses spatiotemporal action localisation and
classification task jointly using a single round of optimisation (cf. Section 6.4). One of the core building blocks of this new architecture is a 3D regional proposal network (RPN) (cf. Section 6.3.3). I designed a 3D-RPN which generates space-time video region hypotheses inplace of frame-level 2D proposals. Unlike in standard action detection approach [2,3,6,11], I implemented a simple but efficient regression technique for regressing such 3D proposals (cf. Section 6.4.1).

The output of this new action detection network is a set of action micro-tubes (cf. Section 1.5.3) instead of a set of 2D detection windows as in [2,3,6,11]. I also proposed a new action tube generation algorithm suitable for connecting these micro-tubes so generated, which exploits the temporal encoding learnt by the network (cf. Section 6.5). I demonstrated quantitatively that our model outperforms state-of-the-art appearance-based models, while being highly competitive with methods which exploit both appearance and motion features (cf. Section 6.7). Moreover, to the best of my knowledge, in the action detection community I am the first to apply “bilinear interpolation” [49,50] (instead of a max-pooling [7]) for ROI (region of interest) feature pooling (cf. Section 6.3.4).

A bilinear interpolation layer allows the gradients of the loss to flow backwards with respect to both the inputs (a) convolutional features and (b) coordinates of bounding boxes. Thus, using bilinear interpolation layer, affine or morphed region proposals can be predicted in place of rectangular windows [51].

Finally, an extension of Chapter 6 is presented (cf. Chapter 7) where the action detection performance and speed are significantly improved by leveraging on optical flow based deep features and a training strategy which incorporates both long and short distance frames. More specifically, I proposed to add an additional optical flow based motion stream to the existing deep architecture in Chapter 6. I demonstrated that by training the network on both long and short distance video frame pairs, and subsequently testing it on long distance pairs can improve the detection performance and speed (cf. Section 7.5). To deal with micro-tubes generated from long distance frame pairs at test-time, I implemented a simple but elegant bounding box interpolation algorithm (cf. Section 7.5.2) which behaves gracefully and makes the tube generation process relatively faster (cf. Section 7.5.3).
1.7 Resulting publications, softwares and media

1.7.1 List of Publications

This PhD thesis has led to the following publications.


1.7.2 List of Softwares

The following softwares from this thesis are available online.

- Source code for our BMVC 2016 work is publicly available online at: https://bitbucket.org/sahasuman/bmvc2016_code. Source code developed using MatCaffe (the Matlab wrapper for Caffe deep learning toolbox).

- Matlab source code for our action instance segmentation work is available online at: https://bitbucket.org/sahasuman/matvis/ (private access).

- Lua and Torch based source code for our AMTnet work is available online at: https://bitbucket.org/sahasuman/amtnet_iccv2017 (private access).

1.7.3 In the media

The following YouTube videos showcasing various qualitative results of this thesis.
• BMVC 2016 work [6] - YouTube demo video link
  \url{https://youtu.be/vBZsTgjhWaQ}.

• Action instance segmentation work [13]:
  (a) YouTube demo video link - main paper
  \url{https://youtu.be/fqqgFQzmkfM}
  (b) YouTube demo video link - generation of optical flow trajectories [52]
  \url{https://youtu.be/iaZ2x1LqFxA}
  (c) YouTube demo video link - generation of supervoxels [53]
  \url{https://youtu.be/skzG4uolcyw}
  (d) YouTube demo video link - human action segmentation [54]
  \url{https://youtu.be/cPjbjAPm2jo}. 
Chapter 2

Related work

In this chapter, we present a literature review on action detection. We first briefly review the prominent work in action classification in Section 2.1. Then in Section 2.2.1, we outline the recent advances in temporal action detection. Finally, we review the state of the art in spatiotemporal action detection in Section 2.2.2.

2.1 Action classification

A plethora of action classification methods have been proposed to recognise human actions in videos. In this section, we briefly present the most noticeable work in action classification. For a detail review, we refer to the recent surveys [55–58]. The major advancements in the field of action classification can be attributed to the rapid progress in video representation, starting from shallow representation [32] to the latest generation deep representation [36]. Shallow representation based methods are mostly handcrafted and relatively simple. Whereas, deep convolutional neural networks (CNNs) are built with multiple layers of non-linear feature extractor and are relatively more sophisticated structure than the standard shallow representation. In the computer vision community, it has been demonstrated that a CNN can learn a better representation by increasing the depth of the network i.e. by increasing the number of layers [48]. Therefore, representation learnt by these networks are often referred as “deep” representation. In contrast, the classical representation is referred as “shallow”.

26
2.1.1 Shallow representation

Most of the shallow representation based methods rely on local spatio-temporal features (video descriptors) where, space-time interest points are detected using either dense fixed grid or a variety of Interest Point Detectors (IPDs) \cite{59,62}, subsequently, local spatio-temporal features are computed from the pixels around each interest point to describe space-time patches. These descriptors are then transformed into more invariant representations using higher order encodings such as bag-of-visual-words (BoVW) or Fischer vectors. Finally, the encoded feature vectors are used to train classifiers (e.g. SVM, decision forests) for action recognition. As this interest point based local feature representation avoids a strict assumption about an action’s global structure, it outperforms global representation where videos have realistic actions captured under uncontrolled settings. The local space-time features are directly computed from raw pixel values, and thus, it minimises the risk of failure cases may arise due to the use of error-prone processing steps such as long-term tracking, contour/silhouette extraction or background/foreground segmentation. Mostly, the local spatio-temporal features were derived from their 2D counterparts: Cuboid \cite{61}, 3D-SIFT \cite{63}, HoG-HoF \cite{64}, Local Trinary Patterns \cite{65}, HoG3D \cite{66}, extended SURF \cite{62}, C2-shape features \cite{67}. For instance, Laptev \cite{60} and Klaser et al. \cite{66} extend Harris cornerness criterion \cite{68} and Histogram of Oriented Gradient (HOG) \cite{69} to videos respectively. Whereas, for video descriptors, Laptev et al. \cite{64} compute both HOG and HOF (Histogram of Optical Flow) features.

Unlike still images, the defining features of video is motion, and another effective way to encode motion is to extract features along trajectories. Thus, a variant of shallow representation is the trajectory based features in which motion in video is represented by long-term point trajectories \cite{70,72}. However, tracking image points over long video sequence is challenging due to several factors such as large displacements, occlusions and above all it is expensive.

To overcome these limitations, Matikainen et al. \cite{73} and Wang et al. \cite{74} track point coordinates for a relatively shorter duration (e.g. 15 frames) and aggregate these short-term trajectories (also known as tracklets or trajectons) for video representation. Unlike long-term, short-term trajectories are robust to drifting (due to their shorter length) and can be directly computed from optical flow \cite{75,76}. Space-time patches (video subvolumes) centred at these short-term trajectories is often described with both appearance and motion features. For instance, One of the most successful video descriptors is the dense trajectory \cite{42} which is formed by combining the HoG-HoF \cite{64} and motion boundary histogram
(MBH) \cite{77} descriptors, together with a sequence of optical flow displacement vectors. Unlike local space-time feature based approaches, dense trajectory features do not rely on interest points and extract features from trajectories composed of points from dense grid. The action recognition performance of these trajectory based approaches can be further improved by discarding the motion caused by the camera movements \cite{78-80}. To this end, camera motion is estimated by matching feature points (e.g. SURF \cite{81}) between video frames. At the same time, a human detector is employed to ignore those matches belong to the actors, to consider the motion associated with the action.

For action classification, several promising methods \cite{42,63,64,66,82} use higher order encodings to transform those aforementioned video descriptors (local spatiotemporal or trajectory based features) into a more invariant representation. One such representation is the well known bag-of-visual-words (BoVW) in which, first a dictionary of visual words are obtained by partitioning the descriptors space into $K$ clusters using $k$-means and subsequently, each video descriptor is associated with its closely matched $k$-th cluster in the dictionary. The close match is computed as per the Euclidean distance. Finally, a video is represented using a bag-of-visual-word histogram $\mathbf{h} = (h_1, \ldots, h_K)^T$, where $h_k$ is the frequency of occurrence of visual word ‘$k$’ in $\mathbf{h}$.

Another popular higher order encoding is the Fisher vector \cite{83} in which, instead of clustering the feature space using $k$-means as done in BoVW approach, it is assumed that the features are distributed as per a Gaussian Mixture Model (GMM) with $K$ components. Unlike the lossy feature quantisation step \cite{84} in BoVW, in Fisher vector encoding, each feature point is associated with a specific Gaussian using a soft assignment, and thus, it encodes additional information about the feature distribution. Moreover, Fisher vector use second-order statistics (i.e. the variance) and achieved superior video classification results \cite{79}. Other prominent work which use higher order encodings are VLAD \cite{85} and super vector coding \cite{86} which make use of the mean of the feature points associated with each visual words.

### 2.1.2 Human location centric approach

Unlike the shallow representation based methods (§ 2.1.1) in which the action’s location information is discarded, there are some approaches \cite{87,88} which make use of this location information. Approaches based on human location is often called as “location centric” or “figure-centric” methods. Efros et al. \cite{87} proposed a method to recognise human actions at a distance (i.e. at a low resolution) using
a combination of both shape and motion (optical flow) features. The action prediction is performed using nearest neighbour techniques. The main limitation is that it requires the actor’s location information beforehand to perform action classification. Blank et al. [88] utilise the action location information by assuming an action instance as a 3D shape extracted by the human silhouettes from the space-time volume. The downside of their approach is the assumption of a known background. Raptis et al. [89] combines ideas from part based models [90, 91] with the extraction of a sparse, low-level video representation to solve for action recognition. Jhuang et al. [9] find that high-level human pose features substantially improve action recognition.

### 2.1.3 Deep representation

Recently, the latest generation of deep representation [36] based methods substantially outperformed the aforementioned shallow representation (§2.1.1) based approaches in several computer vision tasks such as image classification [48,92–94], object detection [1] and semantic segmentation [95] in images. These deep convolutional neural networks (CNNs) have also shown impressive results in video classification. 3D convolution operation is exploited to encode sequence of video frames for classifying videos [96,98]. Ji et al. [96] propose a 3D CNN architecture in which sequence of frames are transformed to 3D feature maps by convolving them with 3D kernels over both spatial and temporal dimensions. Karpathy et al. [97] address the task of large-scale video classification using 1M videos where different deep network architectures are explored by fusing information over consecutive frames at various levels. Tran et al. [98] use $3 \times 3 \times 3$ kernels for all convolution layers to solve for various vision-based tasks including action classification, object recognition, scene classification, action similarity labelling.

Donahue et al. [99] leverage the strengths of deep representation for action classification by combining a CNN with a Long Short-Term Memory (LSTM) where a chunk of video frames (variable length input) are transformed to visual features by processing them through a 2D CNN, and subsequently, these time-varying visual features are fed into a stack of LSTM units to jointly learn both temporal dynamics and convolutional representation of human actions.

One of the most appreciable work in this direction is the “two-stream architecture” proposed by Simonyan and Zisserman [48] in which appearance and motion cues of human actions are encoded using two separate CNNs (streams) by processing RGB and stacked optical flow frames respectively. We adopt this
two-stream structure to design our different deep networks proposed in this work. However, we tackle a different problem (i.e. action detection) than action classification. Also, our appearance and motion fusion strategies are entirely different from [48]. Following the pioneering work of two-stream architecture, Feichtenhofer et al. [40] study various ways of fusing appearance and motion streams to take the utmost advantage of the spatio-temporal information. They show that the new ConvNet architecture based on their findings achieves state-of-the-art results in action classification.

More recently, a few improvements have been proposed to obtain better spatio-temporal deep representation and to speed-up the action recognition task [100–103]. Arandjelovic et al. [100] explicitly model a long-range inhomogeneous video dynamics for accurate recognition of complex human activities. They bring together both the deep feature representation and VLAD encoding [104] to capture short-term and long-range video dynamics. Motivated by the fact that discriminative actions may present sparsely in a few key video subvolumes, and most of the remaining subvolumes may not contain any action, Zhu et al. [101] propose a key volume mining deep framework for action recognition. Bilen et al. [102] introduce a concept of dynamic image network which provides a compact video representation using a ranking machine encoding the temporal evolution of video frames. The most computationally expensive step in the two-stream framework [48] is the optical flow computation. Zhang et al. [103] address this issue by replacing the optical flow CNN with a motion vector CNN. Their new framework can perform at real-time with a recognition speed of 390.7 fps (frames per second) and show comparable classification accuracy to the state-of-the-art.

2.2 Action detection

"Action detection" is also called as "action localisation". In this section we review various work for spatial, temporal and spatiotemporal action detection.

2.2.1 Temporal action detection

The temporal detection of actions [105–106] and gestures [107] in temporally untrimmed videos has also recently attracted much interest [44,108]. In temporal detection, the goal is to locate the optimal subvolume in the 3D video search space. The early approaches to temporal detection rely on sliding-window tech-
Video subsequences with varied temporal extents are uniformly sampled and the one with the maximum classification score is counted as the predicted temporal extent of the action. Unlike spatiotemporal localisation where the search space is extremely large (i.e. it spans over both spatial and temporal dimensions), in temporal detection, the search space is 1-dimensional, and thus, a sliding-window approach is still acceptable.

Gaidon et al. [111] propose a more structured representation which is based on decomposing an action as a sequence of atomic action units (actoms). The downside of their approach is the additional cost required to annotate those actoms. Niebles et al. [112] exploit temporal patterns of human activities and represent activities as composition of motion segments. They make use of Deformable Part Models [91] by inferring temporal scales and anchor points for sub-events of each activity category. Oneata et al. [113] reduce the memory and computational cost by proposing an approximation to the normalized Fisher vector which further enables to replace the exhaustive sliding-window search by a more efficient branch-and-bound search [114]. Richard and Gall [115] propose a probabilistic model to solve for temporal action localisation by jointly modelling the temporal segmentation and segment classification task. For video segment representation, they use Fisher vector of improved dense trajectories and the overall objective (segmentation and classification) is maximised using dynamic programming.

CNNs are capable of reducing frequency variations, whereas, Long Short-Term Memory recurrent neural networks (LSTMs) have shown their ability in temporal modeling (or sequence learning), more specifically, in speech recognition [116] and language translation [117,118]. Driven by these recent success of CNNs in visual representation and LSTMs to effectively encode long- and short-term time-varying signals (speech or video), growing research interests have been noticed to either solve temporal action localisation by purely using CNNs [119] or to combine both CNNs and LSTMs [120,121]. These work mainly focus on tasks such as: improving representation to better capture motion information at multiple scales, exploring temporal consistency, addressing the difficulties associated with expensive sliding-window technique.

Based on the insight that the temporal detection is a process of observing frame glimpses and refining detection hypotheses, Yeung et al. [120] combine backpropagation and reinforcement learning to train a network for temporal action detection. Their framework comprises of a CNN for visual representation
along with a recurrent neural network (RNN) as an agent and avoids an expensive sliding window scheme to sample action proposals. They introduce a reward mechanism which enables the agent to learn a policy - “where to look next and when to emit a prediction”. Yuan et al. [121] address the difficulties in multi-resolution sliding window approach by proposing a descriptor (called as PSDF) which captures the multi-resolution context around anchor frames. For temporal consistency, they further combine the PSDF descriptor with a recurrent neural network (RNN). Shou et al. [119] exploit the effectiveness of deep CNNs for temporal detection by training three separate networks: (a) proposal, (b) classification and (c) localisation CNNs. They use sliding window to generate action proposals at different temporal scales.

2.2.2 Spatio-temporal action detection

Action cuboid hypotheses and sliding-window based approach. Initial attempts for space-time action detection is based on generating a exhaustive set of cuboid shaped action proposals at plausible spatio-temporal scales using a sliding-window technique [122, 123]. The major drawbacks of these methods are, firstly, the assumption that an action instance within a video has a fixed spatial extent (i.e. an action can be localised using a cuboid video subvolume) is not realistic for unconstrained real-world videos due to the fact that spatial location of an action instance may vary over time. Secondly, the video space is much larger than the image space, and thus, a sliding-window scheme is extremely expensive. For instance, consider a video of size $w \times h \times n$, where $w \times h$ is the spatial and $n$ is the temporal extent, the total number of possible 3D action proposal is of $O(w^2h^2n^2)$ [124], it is computationally infeasible to explore such a large search space even for a moderate size video sequence. Lately, Tian et al. [125] perform spatio-temporal action localisation by generating 3D action subvolumes using DPM (deformable part based models) [91], and subsequently, they perform a template matching during test time using a sliding window (i.e. a sliding subvolume) approach. [make sure that Tian et al.’s work outputs a cuboid and not a tube]

Human location centric approach. As mentioned earlier in Section 2.1.2 that approaches centered at human location is often called as “location-centric” approach. Prest et al. [126] use human location information for action detection by first detecting humans and objects in videos, and subsequently, tracking the human-object interactions. Lan et al. [127] treat the location of the actor as a latent variable in the Latent Support Vector Machine framework used in the de-
formable part based model (DPM) \cite{33} and train it to predict both the location of the actor and the class label of the action simultaneously. Tian et al. \cite{125} study the generalization of deformable part based models \cite{33} for action localisation using HoG-3D \cite{66}. Klaser et al. \cite{128} propose a human-centric approach where first, spatio-temporal human tracks are obtained using a human detector and a KLT tracker and then, specific actions are classified within the human tracks using a sliding window HOG-3D descriptor \cite{66}. They reported action detection results for only two actions (phoning and standing up). In contrast, in our experiments, we consider 24, 21 and 10 diverse action categories belong to three different human action detection datasets: UCF-101-24 \cite{8}, J-HMDB-21 \cite{9} and LIRIS HARL \cite{14} respectively. For action detection, Wang et al. \cite{129} use temporal sliding window and the human pose annotations to capture the relations among dynamic-poselets using a sequential skeleton model.

**Unsupervised 3D action proposals and shallow representation based approach.** Unsupervised 2D region proposal generation algorithms (for object detection in images) \cite{37,38,130} have proven their ability to significantly reduce the search complexity (over the exhaustive sliding-window technique). Motivated by this success, 3D counterparts of these 2D proposal algorithms are heavily exploited to generate space-time region proposal to solve for action detection \cite{26,27}. Jain et al. \cite{26} and Oneata et al. \cite{27} extend the unsupervised Selective Search \cite{37} and Prime object proposal \cite{130} algorithms to their 3D counterparts respectively to generate action tubes (cf. Section 1.2) or tubelets. They use shallow representation such as dense trajectories \cite{42} and bag-of-visual-words encoding for action tube classification. More specifically, first, video segmentation is performed to generate supervoxels, and then, supervoxels are merged to form action tubes based on different similarity measures such as color, texture, motion or their size. Similarly, Soomro et al. \cite{43} use video segmentation to generate supervoxels (3D action proposals) and use shallow representation for action classification. More specifically, their model learns contextual relations by capturing displacements between supervoxels belong to foreground and background, every supervoxel is encoded using bag-of-visual-words representation on improved dense trajectory \cite{131} features. A CRF is used to find the action proposals and one-verse-all SVMs are used to score them. Such ‘supervoxels’, however, may end up spanning very long time intervals, failing to localise each action instance individually.

The major drawback is that the video segmentation process is highly expen-
sive and not practical for long duration videos or for applications require online and real-time processing. For instance, it takes several minutes to segment a video clip of resolution $400 \times 720$ with a temporal length of 55 frames [28]. Moreover, supervoxels with longer temporal variance can make the graph over all supervoxels brittle [132]. Further, generic object proposal generation algorithms [37,38,130] are unsupervised in nature and follow a greedy agglomerative clustering of supervoxels, and thus, can not be trained specifically on human action detection datasets. In contrast, we use a fully supervised region proposal generation approach which substantially reduces the computing cost (cf. Chapter 4 & Chapter 6) and can be optimised jointly with the overall action detection objective.

To alleviate the expensive video segmentation process, van Gemert et al. [28] and Yu et al. [30] completely bypass the video segmentation step. van Gemert et al. [28] generate action tubes by clustering the dense trajectories [42] as per their similarity measures defined by the the HoG, HoF and MBH descriptors. However, since their approach completely rely on dense-trajectory features, i.e. both proposal generation and tube classification is based on dense-trajectories, it may not work on actions characterised by small motions. Further, Yu et al. [30] propose an generic action proposal generation method which avoids the expensive video segmentation and shows nearly real-time performance on normal desktop PC. Their methods is based on “actionness” measure [39] and requires localised training samples. Marian Puscas et al. [29] extract unsupervised 3D action proposals (tubes) by leveraging the strengths of both appearance-based static “objectness” (i.e. Selective Search [37]) and motion (dense trajectories [42]) information. They further refine the tubes by applying transductive learning. [Note for me: cross check if Soomro etal. paper does temporal trimming, better read this paper once.] Soomro et al. [133] recently proposed an online method which can predict an actions label and location by observing a relatively smaller portion of the entire video sequence. However, [133] only works on temporally trimmed videos and not in real-time, due to expensive segmentation method.

**Two-stream deep representation based approach.** Indeed methods which exploit the two-stream deep architecture [31] (cf. Section 1.5.6) and temporally connect frame-level region proposals [37,38] for action detection have risen to the forefront of current research. Gkioxari and Malik [2] leverage the deep framework for object detection [1] combined with a two-stream architecture [31] to tackle action detection. Their framework relies on Selective Search proposals [37] and an expensive multi-stage training pipeline as in [1] (cf. Sec-
tion [1.5.1]. However, as the videos used to evaluate their work only contain one action and were already temporally trimmed (J-HMDB-21 [9]), it is not possible to assess their temporal localisation performance. Weinzaepfel et al. [3] also take a similar approach as [2] by adopting a deep object detection framework [1] and a two-stream network structure [31]. They replace the slow Selective Search proposals with relatively faster EdgeBoxes [38] proposals. Further, they use a tracking-by-detection approach based on a novel track-level descriptor called a Spatio-Temporal Motion Histogram. Moreover, [3] achieves temporal trimming using a multi-scale sliding window over each track, making it inefficient for longer video sequences. The common drawbacks of these approaches are: 1) they are computationally expensive and slow due to several factors such as expensive region proposal algorithms, CNN feature extraction and feature caching (cf. Section 1.5.1); 2) they require multi-stage training i.e. frame-level visual representation is learnt by a CNN and action classification is learnt by a set of one-versus-all SVMs; 3) due to unsupervised nature of the proposal generation process, proposal generation task can not be optimised jointly with the overall action detection objective.

We improve on both [2,3] by proposing an elegant and effective solution (cf. Chapter 4). Firstly, we train a deep CNN for proposal generation (cf. Section 4.3.1) i.e. the proposal generation task now can be learnt jointly with the overall action detection objective. Further, computing proposals using a CNN leading to nearly a cost-free solution [7]. Secondly, we train a CNN to learn frame-level action representation, action classification and proposal box regression (cf. Section 4.3.2) which further reduces the computing cost substantially. With this new framework, the expensive feature extraction, feature caching and SVM training steps are completely bypassed. 3) We further, replace the expensive sliding-window for temporal detection [3] with an efficient dynamic programming approach (cf. Section 4.3.4). Some of the reviewed approaches [3,28] could potentially be able to detect co-occurring actions. However, [3] limit their method to produce maximum of two co-occurring detections per class, while [28] does so on the MSRII dataset [123] which only contains three action classes of repetitive nature (clapping, boxing and waving). In contrast, we report action detection results where our proposed framework can detect multiple co-occurring detections per class.

Up till now, we have seen that deep architectures have been increasingly applied of late to action classification [31,96–98], spatial [2], temporal [119] and spatio-temporal [3,6,11] action localisation. While many works concern either
spatial action localisation \[12, 43, 132, 134\] in trimmed videos or temporal localisation \[108, 111, 113, 119, 120, 122, 135\] in untrimmed videos, only a handful number of methods have been proposed to tackle both problems jointly. Spatial action localisation has been mostly addressed using video segmentation based methods \[12, 43, 132\] or by temporally linking frame-level unsupervised region proposals \[2, 3, 134\]. Most recently, supervised frame-level action proposal generation and classification have been used by Saha et al. \[6\] and Peng et al. \[11\], via a Faster R-CNN \[7\] object detector, to generate frame level detections independently for each frame and link them in time in a post-processing step. Unlike \[2, 3, 28\], current methods \[6, 11, 134\] are able to leverage on more faster and elegant deep architectures \[7, 136\] as compared to \[1\] for frame level detection. However, tube construction is still tackled separately from region proposal generation.

The proposed deep network architectures (cf. Chapter \[6\] & Chapter \[7\]) outputs micro-tubes (i.e. the smallest possible spatio-temporal action region in a video) (cf. Section \[1.5.3\]) which span across successive frames, and are labelled using a single soft-max score vector, in opposition to \[2, 3, 6, 11\] which output 2D detection windows at test time. Unlike \[2, 3, 6, 11\], our model is end-to-end trainable and requires a single step of optimisation per training iteration. To the contrary, \[2, 3\] use a multi-stage training strategy mutated from R-CNN object detection \[1\] which requires training two CNNs (appearance and optical-flow) independently, plus a battery of SVMs. Compared to \[2, 3, 6, 11\], which heavily exploit expensive optical flow maps, the proposed action detection framework (Chapter \[6\]) learns spatiotemporal feature encoding directly from raw RGB video frames. Unlike \[2, 3, 6, 11\], which fuse appearance and motion cues at test time, our action detection pipeline (Chapter \[7\]) fuses RGB and flow features at training time.

2.3 Action instance segmentation

Action instance segmentation in video is yet an unexplored research area in computer vision. Considerable amount of research has been done to tackle the problem of object instance segmentation in still images \[137, 142\]. However, we could not find any substantial work addressing the problem of action instance segmentation in videos. Action instance segmentation provides a more elegant and robust solution for accurate human action localisation as compared to a
pure detection approach (cf. Section 1.5.5). Unlike detection, an instance segmentation method can accurately localise actions at a more finer pixel-level by assigning labels to pixels which are both class- and instance-aware.

Early work on object instance segmentation proposed by [137,138]. However, instance segmentation has become a more active research topic after the “Simultaneous Detection and Segmentation” (SDS) work of Hariharan et al. [139]. They detect all instances of an object category in an image and assign a unique instance-aware label to each instance of that category. Their method is based on the R-CNN object detection pipeline [1]. Several methods [140–142] have extended their work [139]. However, none of these aforementioned methods can perform human action instance segmentation in videos and are tailored for object instance segmentation in images. In contrast, we propose a two-stream deep representation based framework which exploit both static appearance and motion information from RGB and optical flow signals respectively, and jointly perform action detection and action instance segmentation in video (cf. Chapter 5). To the best of our knowledge, we are the first to introduce a deep architecture based action instance segmentation framework.
Chapter 3

Datasets and evaluation metric

Need to work on this chapter

From BMVC2016 paper

In order to evaluate our spatio-temporal action detection pipeline we selected what are currently considered among the most challenging action detection datasets: UCF-101 [8], LIRIS HARL D2 [14], and J-HMDB-21 [9]. UCF-101 is the largest, most diverse and challenging dataset to date, and contains realistic sequences with a large variation in camera motion, appearance, human pose, scale, viewpoint, clutter and illumination conditions. Although each video only contains a single action category, it may contain multiple action instances of the same action class. To achieve a broader comparison with the state-of-the-art, we also ran tests on the J-HMDB-21 [9] dataset. The latter is a subset of HMDB-51 [?] with 21 action categories and 928 videos, each containing a single action instance and trimmed to the action’s duration. The reported results were averaged over the three splits of J-HMDB-21. Finally we conducted experiments on the more challenging LIRIS-HARL dataset, which contains 10 action categories, including human-human interactions and human-object interactions (e.g., ‘discussion of two or several people’, and ‘a person types on a keyboard’). In addition to containing multiple space-time actions, some of which occurring concurrently, the dataset contains scenes where relevant human actions take place amidst other irrelevant human motion.

For all datasets we used the exact same evaluation metrics and data splits as in the original papers. In the supplementary material, we further discuss all implementation details, and propose an interesting quantitative comparison between Selective Search- and RPN-generated region proposals.

From CVPR2016 paper

1http://liris.cnrs.fr/voir/activities-dataset
In order to evaluate our multi-class human activity detection algorithm, we selected the challenging LIRIS HARL D2 human activities dataset [14]. The dataset was created for an action detection competition in which 70 teams registered. The large number of action classes for detection compared to previous datasets [15,16] and its difficulty meant that only two teams [143,144] submitted results, to which we compare our results (§ ??). The LIRIS dataset is complex because it contains image sequences containing multiple actions annotated in space and time, some of which occur simultaneously. Moreover, it contains scenes where relevant human actions take place amidst other irrelevant human motion (i.e., other people performing irrelevant actions). The LIRIS dataset contains 10 action categories, which include human-human interactions and human-object interactions, for example, ‘discussion of two or several people’, and ‘a person types on a keyboard’. A full list of categories may be found on the dataset’s website [2]. In particular, we used the D2 sequences shot with a Sony camcorder with a resolution of 720 × 576, and captured at 25 frames per second.

3.0.1 Performance indicators

The qualitative and quantitative performance of our approach was computed using the evaluation tool provided for the LIRIS-HARL competition [14]. Firstly, any detected action tube is assigned to the closest ground truth tube, based on a normalised measure of overlap over all its frames. Secondly, a detected action tube is accepted as positive if detected and ground truth tubes have the same class, and: i) there is sufficient overlap with respect to thresholds on ‘spatial pixel-wise recall’ $t_{sr}$, and ‘temporal frame-wise recall’ $t_{tr}$, and ii) the excess duration is sufficiently small with respect to thresholds for ‘spatial pixel-wise precision’ $t_{sp}$, and ‘temporal frame-wise precision’ $t_{tp}$.

Once the four thresholds $t_{sr}, t_{tr}, t_{sp}$ and $t_{tp}$ are fixed, recall and precision may be calculated in the usual way as: $\text{Recall} = \frac{\# \text{correctly found actions}}{\# \text{actions in ground truth}}$, $\text{Precision} = \frac{\# \text{correctly found actions}}{\# \text{number of found actions}}$. The F1-score combines them as: $F1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$. A final performance measure may be obtained by integrating the F1-score over the range of possible threshold values [145]. Four integrated F1-score values $(I_{sr}, I_{sp}, I_{tr}, I_{tp})$ are first calculated by varying one threshold while setting the others to a small value ($\eta = 0.1$). Then, an overall score is obtained by averaging

\[^2\text{http://liris.cnrs.fr/voir/activities-dataset}\]
the four values:

\[ \text{Integrated Performance} = \frac{I_{sr} + I_{sp} + I_{tr} + I_{tp}}{4} \]  

(3.1)

which is independent from arbitrary thresholds on spatial or temporal overlap [145].

From iccv2017 paper

Datasets. All the experiments are conducted using the following two widely used action detection datasets: a) J-HMDB-21 [9] and b) UCF-101 24-class [8].

J-HMDB-21 is a subset of the relatively larger action classification dataset HMDB-51 [?], and is specifically designed for spatial action detection. It consists of 928 video sequences and 21 different action categories. All video sequences are temporally trimmed as per the action’s duration, and each sequence contains only one action instance. Video duration varies from 15 to 40 frames. Ground-truth bounding boxes for human silhouettes are provided for all 21 classes, and the dataset is divided into 3 train and test splits. For evaluation on J-HMDB-21 we average our results over the 3 splits.

The UCF-101 24-class action detection dataset is a subset of the larger UCF-101 action classification dataset, and comprises 24 action categories and 3207 videos for which spatiotemporal ground-truth annotations are provided. We conduct all our experiments using the first split. Compared to J-HMDB-21, the UCF-101 videos are relatively longer and temporally untrimmed, i.e., action detection is to be performed in both space and time. Video duration ranges between 100 and 1000 video frames.

Note that the THUMOS [106] and ActivityNet [?] datasets are not suitable for spatiotemporal localisation, as they lack bounding box annotation.
Chapter 4

Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

4.1 Introduction

[Fabio, please give a pass to this intro section.]

The main motivating ideas of this chapter are as follows:

- deep features are more discriminative and having better generalising capabilities (on large scale real-world image and video data) than shallow features (cf. Section 1.5) for visual (image & video) representation,

- a single-stage end-to-end trainable deep architecture is a better candidate solution than a multi-stage framework with disjoint optimisation (i.e. a CNN and a set of SVMs are separately trained for action classification) (cf. Section 1.5.1),

- a CNN trained on specific action detection datasets can generate relatively high quality and cost-effective action region hypotheses than unsupervised region proposal algorithms (cf. Section 1.5.1),

- a two-stream hypotheses (cf. Section 1.5.6) is beneficial for action detection and an effective fusion scheme to fuse appearance and motion cues can significantly improve the detection performance,

- posing the inter-frame data association (cf. Section 1.3) and temporal localisation (cf. Section 1.3) as optimisation problems and solving them
using a dynamic programming approach is a more cost effective and elegant solution than solving them using a graph-based video segmentation and sliding window technique (cf. Section 1.5.4).

Motivated by these above ideas, in this chapter we propose a novel action detection framework which, instead of adopting an expensive multi-stage pipeline, takes advantage of the most recent single-stage deep learning architectures for object detection [7], in which a fully convolutional neural network (CNN) is trained for generating frame-level region proposals and another CNN-based deep network is trained for both detecting and classifying those proposals in an end-to-end fashion. Subsequently, the appearance and motion based detections (extracted from the respective trained models) are fused using our new test-time fusion technique. Finally, fused frame-level detections are linked in time to form space-time ‘action tubes’ [2], and then, tubes are temporally trimmed to solve for temporal action localisation. Both temporal linking and localisation problems are solved using two optimisation problems via dynamic programming.

We demonstrate that the proposed action detection pipeline is at least $2 \times$ faster in training and $5 \times$ faster in test time detection speeds as compared to [2,3]. In Section 4.4.9, we compare the computing time requirements during training and testing of our approach with [2,3] on the UCF-101 [8] and J-HMDB-21 [9] datasets. Moreover, our pipeline consistently outperforms previous state-of-the-art results (§ 4.4).

Outline. [follow the style of outline given in Phillipi’s thesis.]

Related publication. The work presented in this chapter has appeared in BMVC 2016 [6].

4.2 Overview of the approach

Our approach is summarised in Figure 4.1. We train two pairs of Region Proposal Networks (RPN) [7] and Fast R-CNN [5] detection networks - one on RGB and another on optical-flow images [2]. For each pipeline, the RPN (b), takes as input a video frame (a), and generates a set of region proposals (c), and their associated ‘actionness’ [39] scores [1]. Next, a Fast R-CNN [7] detection network (d) takes as input the original video frame and a subset of the region proposals generated by the RPN, and outputs a ‘regressed’ detection box and a softmax

\[1\text{A softmax score for a region proposal containing an action or not.}\]
classification score for each input proposal, indicating the probability of an action
class being present within the box. To merge appearance and motion cues, we
fuse (f) the softmax scores from the appearance- and motion-based detection
boxes (e) (§ 4.3.3). We found that this strategy significantly boosts detection
accuracy.

After fusing the set of detections over the entire video, we identify sequences
of frame regions most likely to be associated with a single action tube. Detection
boxes in a tube need to display a high score for the considered action class,
as well as a significant spatial overlap for consecutive detections. Class-specific
action paths (g) spanning the whole video duration are generated via a Viterbi
forward-backward pass (as in [2]). An additional second pass of dynamic pro-
gramming is introduced to take care of temporal detection (h). As a result, our
action tubes are not constrained to span the entire video duration, as in [2]. Fur-
thermore, extracting multiple paths allows our algorithm to account for multiple
coccurring instances of the same action class (see Figure 4.2).

Although it makes use of existing RPN [7] and Fast R-CNN [5] architectures,
this work proposes a radically new approach to spatiotemporal action detection
which brings them together with a novel late fusion approach and an original
action tube generation mechanism to dramatically improve accuracy and detec-
tion speed. Unlike [2, 3], in which appearance and motion information are fused
by combining fc7 features, we follow a late fusion approach [7]. Our novel fu-
sion strategy boosts the confidence scores of the detection boxes based on their
spatial overlaps and their class-specific softmax scores obtained from appearance
and motion based networks (§ 4.3.3). The 2nd pass of dynamic programming,
we introduce for action tube temporal trimming, contributes to a great extent to significantly improve the detection performance (§4.4).

Figure 4.2: Action tube detection in a ‘biking’ video taken from UCF-101 [8]. (a) Side view of the detected action tubes where each colour represents a particular instance. The detection boxes in each frame are linked up to form space-time action tubes. (b) Illustration of the ground-truth temporal duration for comparison. (c) Viewing the video as a 3D volume with selected image frames; notice that we are able to detect multiple action instances in both space and time. (d) Top-down view.

An extensive evaluation on the main action detection datasets demonstrates that our approach significantly outperforms the current state-of-the-art, and is 5 to 10 times faster than the main competitors at detecting actions at test time (§4.4). Thanks to our two-pass action tube generation algorithm, in contrast to most existing action classification [31,42,96,97,131] and localisation [2,3] approaches, our method is capable of detecting and localising multiple co-occurring action instances in temporally untrimmed videos (see Figure 4.2).

4.3 Methodology

As outlined in Figure 4.1, our approach combines a region-proposal network (§4.3.1—Figure 4.1b) with a detection network (§4.3.2—Figure 4.1d), and fuses the outputs (§4.3.3—Figure 4.1f) to generate action tubes (§4.3.4—Figure 4.1g-h). All components are described in detail below.

4.3.1 Region Proposal Network

To generate rectangular action region hypotheses in a video frame we adopt the Region Proposal Network (RPN) approach of [7], which is built on top of the last convolutional layer of the VGG-16 architecture by Simonyan and Zisserman [48]. To generate region proposals, this mini-network slides over the convolutional
feature map outputted by the last layer, processing at each location an $n \times n$ spatial window and mapping it to a lower dimensional feature vector (512-d for VGG-16). The feature vector is then passed to two fully connected layers: a box-regression layer and a box-classification layer.

During training, for each image location, $k$ region proposals (also called ‘anchors’) \cite{7} are generated. We consider those anchors with a high Intersection-over-Union ($\text{IoU}$) with the ground-truth boxes ($\text{IoU} > 0.7$) as positive examples, whilst those with $\text{IoU} < 0.3$ as negatives. Based on these training examples, the network’s objective function is minimised using stochastic gradient descent (SGD), encouraging the prediction of both the probability of an anchor belonging to action or no-action category (a binary classification), and the 4 coordinates of the bounding box.

### 4.3.2 Detection network

For the detection network we use a Fast R-CNN net \cite{5} with a VGG-16 architecture \cite{48}. This takes the RPN-based region proposals (§4.3.1) and regresses a new set of bounding boxes for each action class and associates classification scores. Each RPN-generated region proposal leads to $C$ (number of classes) regressed bounding boxes with corresponding class scores.

Analogously to the RPN component, the detection network is also built upon the convolutional feature map outputted by the last layer of the VGG-16 network. It generates a feature vector for each proposal generated by RPN, which is again fed to two sibling fully-connected layers: a box-regression layer and a box-classification layer. Unlike what happens in RPNs, these layers produce $C$ multi-class softmax scores and refined boxes (one for each action category) for each input region proposal.

**CNN training strategy.** We employ a variation on the training strategy of \cite{7} to train both the RPN and Fast R-CNN networks. Shaoqing et al. \cite{7} suggested a 4-steps ‘alternating training’ algorithm in which in the first 2 steps, a RPN and a Fast R-CNN nets are trained independently, while in the 3\textsuperscript{rd} and 4\textsuperscript{th} steps the two networks are fine-tuned with shared convolutional layers. In practice, we found empirically that the detection accuracy on UCF101 slightly decreases when using shared convolutional features, i.e., when fine tuning the RPN and Fast-RCNN trained models obtained after the first two steps. As a result, we train the RPN and the Fast R-CNN networks independently following only the 1\textsuperscript{st} and 2\textsuperscript{nd} steps of \cite{7}, while neglecting the 3\textsuperscript{rd} and 4\textsuperscript{th} steps suggested by \cite{7}.
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

4.3.3 Fusion of appearance and motion cues

In a work by Redmon et al. [146], the authors combine the outputs from Fast R-CNN and YOLO (You Only Look Once) object detection networks to reduce background detections and improve the overall detection quality. Inspired by their work, we use our motion-based detection network to improve the scores of the appearance-based detection net (cf. Figure 4.1).

Let \(\{b_{si}\}\) and \(\{b_{fj}\}\) denote the sets of detection boxes generated by the appearance- and motion-based detection networks, respectively, on a given test frame and for a specific action class \(c\). Let \(b_{f\text{max}}\) be the motion-based detection box with maximum overlap with a given appearance-based detection box \(b_{si}\). If this maximum overlap, quantified using the IoU, is above a given threshold \(\tau\), we augment the softmax score \(s_c(b_{si})\) of the appearance-based box as follows:

\[
s^*_c(b_{si}) = s_c(b_{si}) + s_c(b_{f\text{max}}) \times \text{IoU}(b_{si}, b_{f\text{max}}). \tag{4.1}
\]

The second term adds to the existing score of the appearance-based detection box a proportion, equal to the amount of overlap, of the motion-based detection score. In our tests we set \(\tau = 0.3\).

4.3.4 Action tube generation

The output of our fusion stage (§ 4.3.3) is, for each video frame, a collection of detection boxes for each action category, together with their associated augmented classification scores (Equation 4.1). Detection boxes can then be linked up in time to identify video regions most likely to be associated with a single action instance, or action tube. Action tubes are connected sequences of detection boxes in time, without interruptions, and unlike those in [2] they are not constrained to span the entire video duration.

They are obtained as solutions to two consecutive energy maximisation problems. First a number of action-specific paths \(p_c = \{b_1, \ldots, b_T\}\), spanning the entire video length, are constructed by linking detection boxes over time in virtue of their class-specific scores and their temporal overlap [Note for Fabio: is it temporal or spatial overlap? I think it should be spatial overlap between two boxes from \(f_t\) and \(f_{t+\Delta}\).]. Second, action paths are temporally trimmed by ensuring that the constituting boxes’ detection scores are consistent with the foreground label \(c\).
Building action paths. We define the energy $E(p_c)$ for a particular path $p_c$ linking up detection boxes for class $c$ across time to be the sum of unary and pairwise potentials:

$$E(p_c) = \sum_{t=1}^{T} s^*_c(b_t) + \lambda_o \sum_{t=2}^{T} \psi_o(b_t, b_{t-1}), \quad (4.2)$$

where $s^*_c(b_t)$ denotes the augmented score (Equation 4.1) of detection $b_t$, the overlap potential $\psi_o(b_t, b_{t-1})$ is the IoU of the two boxes $b_t$ and $b_{t-1}$, and $\lambda_o$ is a scalar parameter weighting the relative importance of the pairwise term. The value of the energy (Equation 4.2) is high for paths whose detection boxes score highly for the particular action category $c$, and for which consecutive detection boxes overlap significantly. We can find the path which maximises the energy,

$$p_c^* = \argmax_{p_c} E(p_c) \quad (4.3)$$

by simply applying the Viterbi algorithm \[2\].

Once an optimal path has been found, we remove all the detection boxes associated with it and recursively seek the next best action path. Extracting multiple paths allows our algorithm to account for multiple co-occurring instances of the same action class.

Smooth path labelling and temporal trimming. As the resulting action-specific paths span the entire video duration, while human actions typically only occupy a fraction of it, temporal trimming becomes necessary. The first pass of dynamic programming (Equation 4.2) aims at extracting connected paths by penalising regions which do not overlap in time. As a result, however, not all detection boxes within a path exhibit strong action-class scores.

The goal here is to assign to every box $b_t \in p_c$ in an action path $p_c$ a binary label $l_t \in \{c, 0\}$ (where zero represents the ‘background’ or ‘no-action’ class), subject to the conditions that the path’s labelling $L_{p_c} = [l_1, l_2, \ldots, l_T']$: i) is consistent with the unary scores (Equation 4.1); and ii) is smooth (no sudden jumps).

As in the previous pass, we may solve for the best labelling by maximising:

$$L^*_{p_c} = \argmax_{L_{p_c}} \left( \sum_{t=1}^{T} s_t(b_t) - \lambda_l \sum_{t=2}^{T} \psi_l(l_t, l_{t-1}) \right), \quad (4.4)$$

where $\lambda_l$ is a scalar parameter weighting the relative importance of the pairwise
term. The pairwise potential $\psi_l$ is defined to be:

$$\psi_l(l_t, l_{t-1}) = \begin{cases} 0 & \text{if } l_t = l_{t-1} \\ \alpha_c & \text{otherwise} \end{cases}$$

(4.5)

where $\alpha_c$ is a class-specific constant parameter which we set by cross validation. As each action category has its own short-term and long-range motion dynamics, cross validating $\alpha_c$ for each action class separately improves the temporal detection accuracy (cf. Section 4.4.7). Equation 4.5 is the standard Potts model which penalises labellings that are not smooth, thus enforcing a piecewise constant solution. Again, we solve (Equation 4.4) using the Viterbi algorithm.

All contiguous subsequences of the retained action paths $p_c$ associated with category label $c$ constitute our action tubes. As a result, one or more distinct action tubes spanning arbitrary temporal intervals may be found in each video for each action class $c$. Finally, each action tube is assigned a global score equal to the mean of the top $k$ augmented class scores (Equation 4.1) of its constituting detection boxes.

4.4 Experimental validation and discussion

In order to evaluate our spatio-temporal action detection pipeline we selected what are currently considered among the most challenging action detection datasets: UCF-101 [8], LIRIS HARL D2 [14], and J-HMDB-21 [9]. We use the following abbreviations to denote three different action detection networks: (1) apnet - appearance detection network, (2) monet - motion detection network and (3) amnet - appearance- and motion-based fused model.

4.4.1 Evaluation metrics

To compare our results on UCF-101 and J-HMDB-21 with the state of the art, we used the same evaluation metric proposed by [3]. For LIRIS HARL, we used the evaluation tool provided by the LIRIS HARL competition [145]. Note that the Area Under the Curve (AUC) on J-HMDB-21 reported by [2] is sensitive to negative detections, as AUC increases when adding many easy negatives, whereas mAP is not affected by easy negatives. The LIRIS HARL evaluation metric [145] requires hyper-parameter optimisation, which is a kind of overhead.

---

2Negatives which have lower detection confidence than all positives.
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

Table 4.1: Quantitative action detection results (mAP) on the UCF-101 dataset.

<table>
<thead>
<tr>
<th>Spatio-temporal overlap threshold $\delta$</th>
<th>0.05</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAP [30]</td>
<td>42.80</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>STMH [3]</td>
<td>54.28</td>
<td>51.68</td>
<td>46.77</td>
<td>37.82</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Our (apnet)</td>
<td>68.74</td>
<td>66.13</td>
<td>56.91</td>
<td>48.28</td>
<td>39.10</td>
<td>30.67</td>
<td>22.77</td>
</tr>
<tr>
<td>Our (monet)</td>
<td>67.04</td>
<td>64.86</td>
<td>57.33</td>
<td>47.45</td>
<td>38.65</td>
<td>28.90</td>
<td>19.49</td>
</tr>
<tr>
<td>Our (amnet)</td>
<td>79.12</td>
<td>76.57</td>
<td>66.75</td>
<td>55.46</td>
<td>46.35</td>
<td>35.86</td>
<td>26.79</td>
</tr>
</tbody>
</table>

apnet - appearance network, monet - motion network, amnet - fused model.

to the whole evaluation process. In contrast, mAP does not require any hyperparameter optimisation, and thus, most suitable for measuring performance of spatio-temporal action detection accuracy. The reported metric on UCF-101 and J-HMDB-21 is the mean Average Precision (mAP) at a threshold $\delta = .2$ for spatio-temporal localisation (UCF-101) and $\delta = .5$ for spatial localisation (J-HMDB-21). We report an mAP and Integrated F1-Score [145] for the LIRISHARL dataset at a threshold of $\delta = .1$.

4.4.2 Performance comparison on UCF-101

Table 4.1 presents the results we obtained on UCF-101, and compares them to the previous state-of-the-art [3,30]. We achieve an mAP of 66.75% compared to 46.77% reported by [3] (a 20% gain), at the standard threshold of $\delta = 0.2$. At a threshold of $\delta = 0.4$ we still get a high score of 46.35%, (comparable to 46.77% [3] at $\delta = 0.2$). Note that we are the first to report results on UCF-101 up to $\delta = .6$, attesting to the robustness of our approach to more accurate localisation requirements. Although our separate appearance- and motion-based detection pipelines already outperform the state-of-the-art (Table 4.1), their combination ($\S$ ??) delivers a significant performance increase.

Some representative example results from UCF-101 are shown in Figure 4.3. Our method can detect several (more than 2) action instances concurrently, as shown in Figure ??, in which three concurrent instances and in total six action instances are detected correctly. Quantitatively, we report class-specific video AP (average precision in %) of 88.0, 83.0 and 62.5 on the UCF-101 action categories ‘Fencing’, ‘SalsaSpin’ and ‘IceDancing’, respectively, which all concern multiple inherently co-occurring action instances. Class-specific video APs on UCF-101 are reported in Section 4.4.6.
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

Figure 4.3: Action detection/localisation results on UCF101. Ground-truth boxes are in green, detection boxes in red. The top row shows correct detections, the bottom one contains examples of more mixed results. In the last frame, 3 out of 4 ‘Fencing’ instances are nevertheless correctly detected.

4.4.3 Performance comparison on J-HMDB-21

The results we obtained on J-HMDB-21 are presented in Table 4.2. Our method again outperforms the state-of-the-art, with an mAP increase of 18% and 11% at $\delta = .5$ as compared to [2] and [3], respectively. Note that our motion-based detection pipeline alone exhibits superior results, and when combined with appearance-based detections leads to a further improvement of 4% at $\delta = .5$.

These results attest to the high precision of the detections - a large portion of the detection boxes have high IoU overlap with the ground-truth boxes, a feature due to the superior quality of RPN-based region proposals as opposed to Selective Search’s (a direct comparison is provided in Section 4.4.5). Sample detections on J-HMDB-21 are shown in Figure 4.4. Also, we list our classification accuracy results on J-HMDB-21 in Table 4.3, where it can be seen that our method achieves an 8% gain compared to [2].

Figure 4.4: Sample space-time action localisation results on JHMDB. Left-most three frames: accurate detection examples. Right-most three frames: mis-detection examples.

4.4.4 Performance comparison on LIRIS-HARL

LIRIS HARL allows us to demonstrate the efficacy of our approach on temporally untrimmed videos with co-occurring actions. For this purpose we use LIRIS-HARL’s specific evaluation tool - the quantitative results are shown in
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

Table 4.2: Quantitative action detection results (mAP) on the J-HMDB-21 dataset.

<table>
<thead>
<tr>
<th>Spatio-temporal overlap threshold $\delta$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionTube [2]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>53.3</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Wang et al. [134]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>56.4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>STMH [3]</td>
<td>–</td>
<td>63.1</td>
<td>63.5</td>
<td>62.2</td>
<td>60.7</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Our (apnet)</td>
<td>52.99</td>
<td>52.94</td>
<td>52.57</td>
<td>52.22</td>
<td>51.34</td>
<td>49.55</td>
<td>45.65</td>
</tr>
<tr>
<td>Our (monet)</td>
<td>69.63</td>
<td>69.59</td>
<td>69.49</td>
<td>69.00</td>
<td>67.90</td>
<td>65.25</td>
<td>54.35</td>
</tr>
<tr>
<td><strong>Our (amnet)</strong></td>
<td><strong>72.65</strong></td>
<td><strong>72.63</strong></td>
<td><strong>72.59</strong></td>
<td><strong>72.24</strong></td>
<td><strong>71.50</strong></td>
<td><strong>68.73</strong></td>
<td><strong>56.57</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>56.6</td>
<td>61</td>
<td>62.5</td>
<td><strong>70.0</strong></td>
</tr>
</tbody>
</table>

Table 4.3: Classification accuracy on the J-HMDB-21 dataset.

Table 4.4 and 4.5 and the qualitative results can be visualised in Figure 4.5. Our results are compared with those of i) VPULABUAM-13 [143] and ii) IACAS-51 [144] from the original LIRIS HARL detection challenge. In this case, our method outperforms the competitors by an even larger margin. We report space-time detection results by fixing the threshold quality level to 10% for the four thresholds [145] and measuring temporal precision and recall along with spatial precision and recall, to produce an integrated score. We refer the readers to [145] for more details on LIRIS HARL’s evaluation metrics.

Figure 4.5: Frames from the space-time action detection results on LIRIS-HARL, some of which include single actions involving more than one person like ‘handshaking’ and ‘discussion’. Left-most three frames: accurate detection examples. Right-most three frames: mis-detection examples.

We also report in Table 4.6 the mAP scores obtained by the appearance, motion and the fusion detection models, respectively (note that there is no prior state of the art to report in this case). Again, we can observe an improvement of 7% mAP at $\delta = .2$ due to our fusion strategy. To demonstrate the advantage of our 2nd pass of DP (§ ??), we also generate results (mAP) using only the first DP pass (§ ??). Without the 2nd pass performance decreases by 20%, highlighting the importance of temporal trimming in the construction of action tubes.
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

Table 4.4: Quantitative action detection results on the LIRIS-HARL dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Recall-10</th>
<th>Precision-10</th>
<th>F1-Score-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPULABUAM-13-IQ</td>
<td>0.04</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td>IACAS-51-IQ</td>
<td>0.03</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>(Ours)</td>
<td>0.568</td>
<td>0.595</td>
<td>0.581</td>
</tr>
</tbody>
</table>

Table 4.5: Quantitative action detection results on the LIRIS-HARL dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>$I_{sv}$</th>
<th>$I_{sp}$</th>
<th>$I_{tr}$</th>
<th>$I_{tp}$</th>
<th>IQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPULABUAM-13-IQ</td>
<td>0.02</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>IACAS-51-IQ</td>
<td>0.01</td>
<td>0.01</td>
<td>0.03</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td>(Ours)</td>
<td>0.5383</td>
<td>0.3402</td>
<td>0.4802</td>
<td>0.4739</td>
<td>0.458</td>
</tr>
</tbody>
</table>

$I_{sv}$ - integrated spatial recall, $I_{sp}$ - integrated spatial precision, $I_{tr}$ - integrated temporal recall, $I_{tp}$ - integrated temporal precision, IQ - integrated quality or performance measure [145].

Table 4.6: Quantitative action detection results (mAP) on LIRIS-HARL for different $\delta$.

<table>
<thead>
<tr>
<th>Spatio-temporal overlap threshold $\delta$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance detection model</td>
<td>46.21</td>
<td>41.94</td>
<td>31.38</td>
<td>25.22</td>
<td>20.43</td>
</tr>
<tr>
<td>Motion detection model</td>
<td>52.76</td>
<td>46.58</td>
<td>35.54</td>
<td>26.11</td>
<td>19.28</td>
</tr>
<tr>
<td>Appearance+motion fusion with one DP pass</td>
<td>38.1</td>
<td>29.46</td>
<td>23.58</td>
<td>14.54</td>
<td>9.59</td>
</tr>
<tr>
<td><strong>Appearance+motion fusion with two DP passes</strong></td>
<td><strong>54.18</strong></td>
<td><strong>49.10</strong></td>
<td><strong>35.91</strong></td>
<td><strong>28.03</strong></td>
<td><strong>21.36</strong></td>
</tr>
</tbody>
</table>

4.4.5 Comparative analysis of region proposal quality

Firstly we analysed the quality of Selective Search vs RPN-based region proposals using the Recall-to-IoU measure [7]. We extracted Selective Search (SS) boxes (approximately 1000 boxes/frame) and RPN-based detection boxes (300 boxes/frame) from our detection network on UCF-101 testsplit-1. Also, we applied a constraint on the RPN-based proposals by putting a threshold to their class-specific softmax probability scores $s_c$ and only considering those proposals with $s_c \geq 0.2$. For each UCF-101 action category, we computed the recall of these proposals at different threshold values. Even with a relatively smaller number of proposals and the additional constraint on the classification probability score, RPN-based proposals exhibit much better recall values than SS-based boxes as depicted in Figure 4.6.

4.4.6 Ablation study

We are the first to report an ablation study of the spatio-temporal action localisation performance on UCF-101 dataset. Table 4.7 shows the class-specific video AP (average precision in %) for each action category of UCF-101 gener-
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

Recall-RPN boxes

0.4 0.6 0.8 1
0
0.2
0.4
0.6
0.8
1
0.4 0.6 0.8 1
0
0.2
0.4
0.6
0.8
1
0.4 0.6 0.8 1
0
0.2
0.4
0.6
0.8
1

Recall-SS boxes

0.4 0.6 0.8 1
0
0.2
0.4
0.6
0.8
1
0.4 0.6 0.8 1
0
0.2
0.4
0.6
0.8
1
0.4 0.6 0.8 1
0
0.2
0.4
0.6
0.8
1

Figure 4.6: Performance comparison between Selective Search (SS) and RPN-based region proposals on four groups of action classes (vertical columns) in UCF-101. Top row: recall vs. IoU curve for SS. Bottom row: results for RPN-based region proposals.

Table 4.7: An ablation study of the spatio-temporal detection results (video APs in %) on UCF-101.

<table>
<thead>
<tr>
<th>Actions</th>
<th>Basketball</th>
<th>BasketballDunk</th>
<th>Biking</th>
<th>CliffDiving</th>
<th>CricketBowling</th>
<th>Diving</th>
</tr>
</thead>
<tbody>
<tr>
<td>appearance</td>
<td>30.5</td>
<td>22.7</td>
<td>56.1</td>
<td>44.2</td>
<td>11.5</td>
<td>89.7</td>
</tr>
<tr>
<td>motion</td>
<td>22.9</td>
<td>41.5</td>
<td>52.0</td>
<td>64.6</td>
<td>30.2</td>
<td>86.7</td>
</tr>
<tr>
<td>appearance+motion</td>
<td>36.7</td>
<td>48.3</td>
<td>60.4</td>
<td>73.2</td>
<td>19.9</td>
<td>96.6</td>
</tr>
<tr>
<td>GolfSwing</td>
<td>59.9</td>
<td>95.4</td>
<td>59.2</td>
<td>41.5</td>
<td>48.9</td>
<td>77.8</td>
</tr>
<tr>
<td>HorseDiving</td>
<td>47.0</td>
<td>91.5</td>
<td>62.0</td>
<td>68.3</td>
<td>51.9</td>
<td>88.2</td>
</tr>
<tr>
<td>appearance+motion</td>
<td>66.5</td>
<td>94.1</td>
<td>62.5</td>
<td>55.7</td>
<td>72.6</td>
<td>89.6</td>
</tr>
<tr>
<td>IceDancing</td>
<td>51.8</td>
<td>61.6</td>
<td>87.6</td>
<td>42.7</td>
<td>80.6</td>
<td>31.1</td>
</tr>
<tr>
<td>LongJump</td>
<td>78.9</td>
<td>92.8</td>
<td>86.4</td>
<td>61.3</td>
<td>32.6</td>
<td>51.3</td>
</tr>
<tr>
<td>appearance+motion</td>
<td>86.9</td>
<td>93.8</td>
<td>52.4</td>
<td>76.5</td>
<td>13.2</td>
<td>73.5</td>
</tr>
<tr>
<td>Skiing</td>
<td>83.4</td>
<td>80.0</td>
<td>83.0</td>
<td>67.0</td>
<td>67.3</td>
<td>63.8</td>
</tr>
<tr>
<td>appearance+motion</td>
<td>88.0</td>
<td>99.7</td>
<td>57.5</td>
<td>85.0</td>
<td>15.9</td>
<td>75.6</td>
</tr>
</tbody>
</table>

ated by the appearance- and motion-based detection networks separately, and by the appearance+motion fusion model. Results are generated at a spatio-temporal overlap threshold of $\delta = 0.2$. For 18 out of 24 action classes, our appearance+motion fusion technique gives the best APs. The appearance-based detection net alone achieves the best APs for two classes: HorseRiding (HR) and TennisSwing (TS), while the motion-based detection net outperforms for action classes: CricketBowling (CB), LongJump (LJ), SalsaSpin (SaS) and SoccerJuggling (SJ). It is worth noting that for action classes HR and TS, static appearance cues such as “horse” and “tennis player” are the most discriminative features whereas, for action classes CB, LJ, SaS and SJ, the motion’s temporal dynamics seems to be most discriminative. This could explain the highest APs of appearance- and motion-based networks for these specific actions.
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

Table 4.8: Spatio-temporal detection results (mAP) on UCF-101 using two different sets of $\alpha_c$ values.

<table>
<thead>
<tr>
<th>$\alpha_c$</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_c = 0$</td>
<td>60.77</td>
</tr>
<tr>
<td>class-specific $\alpha_c$</td>
<td>66.75</td>
</tr>
</tbody>
</table>

Table 4.9: Class specific $\alpha_c$ values for each action category in UCF-101 obtained from cross validation.

<table>
<thead>
<tr>
<th>Actions</th>
<th>class-specific $\alpha_c$</th>
<th>BasketballDunk</th>
<th>Biking</th>
<th>CliffDiving</th>
<th>CricketBowling</th>
<th>Diving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basketball</td>
<td>0</td>
<td>0.8</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BasketballDunk</td>
<td>0.2</td>
<td>4</td>
<td>18</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>Biking</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>CliffDiving</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>CricketBowling</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>Diving</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

4.4.7 Impact of label smoothing on detection performance

We conducted experiments to show the significance of the path label smoothing step. More specifically, we show that the class-specific $\alpha_c$ values help to smooth the action paths for each action category independently resulting an overall performance boost in the spatio-temporal detection accuracy. First, we generated detection results on UCF-101 test set (split-1) by setting the constant parameter $\alpha_c = 0$ for each action category. Then, we use the cross validated class-specific $\alpha_c$ values and again generated detection results. In our experiment, we set the spatio-temporal IoU threshold $\delta = 0.2$. Table 4.8 presents the results for both the cases: detection result obtained by setting $\alpha_c = 0$ for each action and result generated using the cross validated class-specific $\alpha_c$ values. Table 4.9 shows the class-specific $\alpha_c$ values obtained by cross validation. Notice that the class-specific $\alpha_c$ improves the detection accuracy (mAP) by 6%. We empirically observed that the class-specific softmax probability scores (from detection network) are not always stable throughout an action path generated by the 1st pass of DP algorithm, i.e., there are sudden jumps in the scores causing a valid action path to be broken by the 2nd pass DP algorithm. The class-specific $\alpha_c$ value helps to stabilise an action path by introducing a certain penalty in the 2nd pass of DP. Due to the fact that each action category has its own temporal duration and speed, different alpha values for different action classes is better than having a single alpha value assigned for all classes.
4.4.8 Additional qualitative detection results

Figure 5.10 provides additional evidence on the temporal detection and spatial localisation performance of our method. The additional results can be seen in the video submitted alongside this document.

![Sample qualitative spatio-temporal localisation results on UCF-101. Each row represents a UCF-101 test video clip. Ground-truth bounding boxes are in green, detection boxes in red.](image)

4.4.9 Computing time analysis for training and testing

We compared detection speed at test time of the combined region proposal generation and CNN feature extraction approach used in (2, 3) to our neural-net based, single stage action proposal and classification pipeline on the J-HMDB-21
Chapter 4. Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos

We found our method to be $10 \times$ faster than [2] and $5 \times$ faster than [3], with a mean of 113.52 [2], 52.23 [3] and 10.89 (ours) seconds per video, averaged over all the videos in J-HMDB-21 split1.

We also performed an analysis of training and testing time requirements of our method in comparison with our main competitors [2,3]. Note that [3] modifies the pipeline of ActionTube [2] by adding a ‘tracking by detection’ module - thus in Table 4.10 and 4.11 while comparing the computation time, we only consider those components of the detection pipelines which are common to both [2] and [3].

**Comparison on UCF-101 dataset.** The comparison is run on the UCF-101 dataset, using 7 NVIDIA Titan X GPUs. Time is computed assuming that appearance- and motion-based CNNs are trained in parallel. Our method is at least $2 \times$ faster in training and $20 \times$ faster in testing on UCF101 (refer Table 4.10). The most time consuming step in [2,3] is CNN feature extraction, as CNN features are extracted for each region proposal and for each video frame, and each feature extraction process requires to run a CNN forward pass. For example, using ActionTube [2]’s approach, for UCF-101’s 240k training video frames with approximately 100 Selective Search based region proposals per frame, we need $240k \times 100$ CNN forward passes to extract features there. In contrast an RPN net needs only $240k$ CNN forward passes, as it uses a single shared convolutional feature map for proposal generation and requires only one CNN forward pass per video frame.

Even in our pipeline RPN region proposal extraction is time consuming. A RPN model takes 100ms to process each frame - multiplied by 240k UCF-101 training video frames, the entire process takes 7 hours. We significantly reduce this time to $\sim 26$ minutes by employing 7 NVIDIA Titan X GPUs in parallel to extract region proposals. Time computation for the competing methods is reported considering 40k training iterations for CNN fine-tuning; for RPN and Fast R-CNN training 320k CNN training iterations are used. Testing time performances for the proposed method are once again reported while using 7 Titan X GPUs in parallel.

**Test-time detection speed comparison on J-HMDB-21.** We compare the video-level detection time of our proposed pipeline with the state-of-the-art [2,3] which use an expensive multi-stage classification strategy. We report comparison results on J-HMDB-21 dataset. We exclude our 2nd pass DP step
Chapter 4. *Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos*

Table 4.10: Training and test time detection speed comparison on UCF-101 with [2, 3].

<table>
<thead>
<tr>
<th>Training time: time computed on 2293 UCF-101 training video clips (split-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-tuning CNNs</td>
</tr>
<tr>
<td>CNN feature extraction</td>
</tr>
<tr>
<td>One vs rest SVMs training</td>
</tr>
<tr>
<td><strong>Total training time required</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test time: time computed on 914 UCF-101 test video clips (split-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN feature extraction</td>
</tr>
<tr>
<td>Fast R-CNN detections</td>
</tr>
<tr>
<td>1st pass DP</td>
</tr>
<tr>
<td>2nd pass DP</td>
</tr>
<tr>
<td><strong>Total test time required</strong></td>
</tr>
</tbody>
</table>

Table 4.11: Test time detection speed comparison on J-HMDB-21 with [2, 3].

<table>
<thead>
<tr>
<th>ActionTube [2], STMH [3]</th>
<th>Average time (Sec./video)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN feature extraction</td>
<td>45.42</td>
</tr>
<tr>
<td><strong>Avg. detection time</strong></td>
<td>113.52 [2] / 52.23 [3]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ours</th>
<th>Average time(Sec./video)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPN proposal generation</td>
<td>4.08</td>
</tr>
<tr>
<td>Detection network</td>
<td>6.81</td>
</tr>
<tr>
<td><strong>Avg. detection time</strong></td>
<td>10.89</td>
</tr>
</tbody>
</table>

due to the fact that J-HMDB-21 video clips do not require temporal trimming. Our 1st pass DP and optical flow based ‘motion frame’ generation steps are common to [2] and our pipeline, and thus, we exclude these steps as well in our comparison. We compare the computation times required for the region proposal generation and CNN feature extraction steps of [2, 3] with our RPN and detection nets computation times. Table 4.11 shows the time required for each step. The reported computation time is averaged over all the videos in the J-HMDB-21 test split1. The time is in second per video clip. All the Experimental results were generated using a desktop computer with an Intel Xeon CPU@3.20GHz and NVIDIA Titan X GPU. Our method is at least 10× faster than [2] and 5× than [3] in detecting actions in a video.
4.4.10 Discussion

The superior performance of the proposed method is due to a number of reasons. 1) Instead of using unsupervised region proposal algorithms as in [37, 38], our pipeline takes advantage of a supervised RPN-based region proposal approach which exhibits better recall values than [37] (cf. Section 4.4.5). 2) Our fusion technique improves the mAPs (over the individual appearance or motion models) by 9.4%, 3.6% and 2.5% on the UCF-101, J-HMDB-21 and LIRIS HARL datasets respectively. We are the first to report an ablation study (cf. Section 4.4.6) where it is shown that the proposed fusion strategy (§4.3.3) improves the class-specific video APs of UCF-101 action classes. 3) Our original 2nd pass of DP is responsible for significant improvements in mAP by 20% on LIRIS HARL (cf. Section 4.4.4) and 6% on UCF-101 (cf. Section 4.4.7). Additional qualitative results are provided in the supplementary video §3 and on the project web page §4 where the code has also been made available.

4.5 Implementation details

4.5.1 Generating correct ground-truth annotations

We downloaded the original XML annotation files available online at: http://crcv.ucf.edu/data/UCF101.php and parsed them using MATLAB XML parser. We found a mismatch between the number of actual ground truth action tubes present in the XML files and the number of tubes available in the annotations of [3, 28]. [3, 28]’s annotations did not contain all the ground truth action tubes. Subsequently, we developed a MATLAB based parser which takes the annotation XML files as input and generates a MATLAB structure which contains all the valid ground truth action tubes both for train and test sets. All the results on UCF-101-24 dataset in this dissertation were generated using our new corrected ground truth annotation which is available online at https://bitbucket.org/sahasuman/bmvc2016_code. Following to our corrections, [11, 28] corrected their action detection results on UCF-101-24 dataset.

4.5.2 Modifications in the existing codebase

We downloaded the publicly available Faster R-CNN MATLAB code from https://github.com/ShaoqingRen/faster_rcnn to train the RPN and Fast R-CNN.
networks. We practically experienced a shortage of RAM memory while training UCF-101 using this code. The original MATLAB code tries to load the entire training data into RAM. For datasets such as UCF-101 the amount of training data is substantial, causing out-of-memory issues. For example, in UCF-101, we have 240k training video frames, a horizontal flipping process for each video frame gives us in total 480k training frames. Loading RPN training data for 480k frames takes more than 64GB of RAM in our experiments. The situation becomes worse in Fast R-CNN training, when the code tries to load training data for 480k × 2000 region proposals which exhausts the entire 128GB of RAM completely. In the default setting, a RPN net takes as input 1 video frame per training iteration and a Fast R-CNN takes as input 2 frames per iteration. Thus, loading the entire training data into RAM can be easily avoided by caching the frame-level training data into disk storage and fetching them as and when required by the CNN training module. We modified the existing MATLAB code to require a smaller amount of RAM memory for both RPN and Fast R-CNN training.

4.5.3 Optical flow based video frame generation

We capture the local motion patterns (of human actions) between two consecutive video frames by computing dense optical flow fields (cf. Section 1.5.7). We compute dense optical flow between each pair of consecutive video frames using the state-of-the-art algorithm in [47]. The 3-channel optical flow values (i.e., flow-x, flow-y and the flow magnitude) were then used to construct ‘motion frames’ [2]. These motion (flow) frames were used to train the motion-based RPN and Fast R-CNN networks.

4.5.4 Selective Search region proposals

In Section 4.4.5 and 4.4.9 we presented a comparative analysis of region proposal quality and train and test time detection speed comparison with the state-of-the-art. In these experiments, we used the Selective Search algorithm to extract region proposals on UCF-101 video frames. We extracted Selective Search (SS) region proposals using the publicly available code from https://github.com/rbgirshick/rcnn. We used the SS’s ‘fast mode’ and obtained approximately 1000 SS boxes per video frame, subsequently, we filtered these SS boxes using the motion saliency scores of [2] and retain on average 100 SS boxes per frame.
4.5.5 Network training and testing

Training data preparation We divided the UCF-101 train split one into two subsets. The first subset consists of 70% (1605 videos ~ 240k frames) and the second subset contains 30% (688 videos) of the training videos from UCF-101 train split one. We selected the videos uniformly at random for each action class and trained the RPN and Fast R-CNN networks using the first subset, while the second subset was used as a validation set for CNN training. For J-HMDB-21 and LIRIS HARL D2 datasets, we used the original training sets provided by the authors [9,145].

CNN weight initialisation The RPN and Fast R-CNN networks [7] were initialised with weights from a pre-trained ImageNet model [147].

CNN solver configuration setting. For UCF-101, we trained both RPN and Fast R-CNN for 320k iterations. For the first 240k iterations we used a learning rate 0.001, while for the remaining 80k iterations a learning rate of 0.0001 was set. For both the J-HMDB-21 and the LIRIS-HARL datasets, we trained both RPN and Fast R-CNN networks for 180k iterations. For the first 120k iterations a learning rate of 0.001 was used - for the remaining 60k iterations, we set the learning rate to 0.0001. The momentum was set to a constant value of 0.9, while weight decay was fixed to 0.0005.

Stochastic Gradient Descent mini-batch size. We selected an SGD mini-batch size of 256 for RPN, and 128 for Fast R-CNN training.

CNN training. First we trained an RPN network with either a set of RGB or optical flow based training video frames. At each training iteration, the RPN takes as input a video frame and its associated ground-truth bounding boxes. Once the RPN net was trained, we used the trained model to extract frame-level region proposals. A trained RPN net outputs a set of region proposals (around 16k to 17k) per frame and their associated actionness scores. We then filtered these region proposals using non-maximal suppression (NMS) and selected top 2k proposals based on their actionness scores. These top 2k region proposals along with the frame and its ground-truth boxes were then passed to a Fast R-CNN for training.
CNN testing. Once training both RPN and Fast R-CNN networks, we extracted region proposals from test video frames using the learnt RPN model. Similarly to what done in the training stage, we filtered the region proposals using NMS - however, at test time, we chose the top 300 region proposals and passed them to the Fast R-CNN network to obtain the final detection boxes: a set of $300 \times C$ regressed boxes and their associated softmax probability scores (where $C$ is the number of ground-truth action categories in a given dataset). For each action category, we first filtered the detection boxes using NMS and then selected the top 5 boxes per frame based on their softmax probability scores. We used an NMS threshold of 0.7 to filter the RPN-generated region proposals, and a threshold of 0.3 when filtering the Fast R-CNN detection boxes.
Chapter 5

Spatio-temporal Action Instance Segmentation and Localisation

5.1 Introduction

[Fabio, please give a pass to this intro section.]

The action detection framework presented in the previous chapter (Chapter 4) along with other competing approaches [2,3,6,11] address the problem of action detection in a setting where videos contain single action category and most of them are temporally trimmed. In contrast, this chapter addresses the problems of both spatio-temporal action instance segmentation (cf. Section 1.5.5) and action detection. Here, we consider real-world scenarios where videos often contain co-occurring action instances belong to different action categories.

Consider the example shown in Figure 5.1, where our proposed model performs action instance segmentation and detection of two co-occurring actions “leaving bag unattended” and “handshaking” which have different spatial and temporal extents within the given video sequence. The video is taken from the LIRIS-HARL dataset.

Outline. [coming soon...]

5.2 Overview of the approach

An overview of the algorithm is depicted in Figure 5.2. At test time, we start by computing binary human motion segmentation (a) for each input video frame, followed by a frame-level region proposal generation (b) (§ 5.3.1). Proposal bounding boxes are then used to crop patches from both RGB and optical flow...
frames (c). Crop image patches are resized to a fixed dimension and fed as inputs to an appearance- and motion-based detection network (d) (§5.3.2) to compute CNN fc7 features. Subsequently, these appearance- and motion-based fc7 features are fused, and later, these fused features are classified by a set of one-vs-all SVMs. Each fused feature vector is a high-level image representation of its corresponding wrapped region and encodes both static appearance (e.g., boundaries, corners, object shapes) and motion pattern of human actions (if there is any). Finally, the top $k$ frame-level detections (regions with high classification scores) are temporally linked in time to build class-specific action tubes (e) and then, these tubes are trimmed to solve for temporal action localisation (§5.3.5). Pixels belong to each action tube are assigned class- and instance-aware action labels by taking advantage of both tube’s class score and the binary action segmentation maps computed in (a). At train time, first action region hypotheses are generated for RGB video frames using Selective Search [37] (§5.3.1), then, pre-trained appearance and motion CNNs (d) are fine-tuned on the wrapped regions extracted from both RGB and flow frames. Subsequently, fine-tuned appearance and motion CNNs are used to compute fc7 features from both RGB and flow training frames, features are then fused and pass as inputs to a set of one-vs-all SVMs for training. A detailed descriptions of these above steps are presented in Section 5.3.

5.3 Methodology

5.3.1 Region proposal generation
Proposals based on motion segmentation. The human motion segmentation algorithm generates binary segmentation of human actions (Figure 5.2 (a)). It extracts human motion from video using long term trajectories [52]. In order to detect static human body parts which don’t carry any motion but are still significant in the context of the whole action, it attaches scores to these regions using a human shape prior from a deformable part-based (DPM) model [91]. By striking balance between the human motion and static human-body appearance information, it generates binary silhouettes of human
actions in space and time. At test time our region proposal algorithm accepts the binary segmented images produced by \cite{[12]}, and generates region proposal hypotheses using all possible combinations of 2D connected components \((2^N - 1)\) present in the binary map (Figure 5.2 (b)), where \(N\) is the number of 2D connected components present in each video frame (cf. Section A.1). In the following subsection, we briefly introduce the human motion segmentation pipeline.

![Figure 5.3:](a) Three sample input video frames showing a “handshaking” action from a test video clip of LIRIS HARL dataset \cite{[x]}. (b) The corresponding motion saliency response generated using long term trajectories \cite{[y]} are shown for these three frames. Notice, the motion saliency is relatively higher for the person at the left, who first enters into the room and then approaches towards the person in the right for “handshaking”. Also note that, motion saliency is computed on the entire video clip, for the sake of visualization, we pick three sample frames.

**Human motion segmentation.** The human motion segmentation algorithm takes as input a sequence of RGB video frames (which contain human action) and outputs binary-labelled space-time video segments where pixels belong to an human action are labelled as foreground and remaining are as background. Firstly, in order to localise and rank “actionness” \cite{[39]}, a human motion saliency feature is computed by exploiting the foreground motion and human appearance information. Foreground motion is estimated by forming a camera model using long term trajectories \cite{[52]} (Figure 5.3) and human appearance based saliency map is generated using a DPM person detector \cite{[91]} (Figure 5.4 (a-c)) trained on
PASCAL VOC 2007\cite{148}. Secondly, to segment human actions, a hierarchical graph-based video segmentation algorithm\cite{149} is used to extract supervoxels at different level of pixel granularity (i.e. different levels of segmentation hierarchy) (Figure 5.5). The foreground motion and human appearance based saliency features are then encoded in the hierarchy of supervoxels using a hierarchical Markov Random Field (MRF) model. This encoding gives the unary potential components. To avoid a brittle graph due to a large number of supervoxels\cite{132}, the MRF graph is built with a smaller subset of supervoxels which are highly likely to contain human actions. Thus, a candidate edge is built between two neighbouring supervoxels based on their optical flow directions and overlaps with a person detection. In the MRF graph structure, supervoxels are nodes and an edge between two supervoxels are built if: (a) they are temporal neighbours i.e. neighbours in the direction of optical flow, or (b) spatial neighbours, i.e. both the supervoxels have high overlaps with a DPM person detection where the person detection has a confidence greater than a threshold. The temporal supervoxel neighbours and the appearance-aware spatial neighbours (Figure 5.4 (d) & (e)) give the pairwise potential components. To avoid leaks and encourage better semantic information, supervoxels (constrained by appearance and motion cues) from higher levels in the hierarchy (Figure 5.5) are supported by the higher-order potential. Finally, the energy of the MRF is minimised using the $\alpha$-expansion algorithm\cite{150,151} and GMM estimation is used to automatically learn the
Figure 5.5: (a) Three sample input video frames showing a “handshaking” action from a test video clip of LIRIS HARL dataset \cite{dataset}. (b) The hierarchical graph based video segmentation results (at three different levels of hierarchy) are shown for these three frames. The three rows show segmentation results for hierarchy level 1, 5 and 10 respectively where 1 is the lowest level with supervoxels having smaller spatial extents and 10 is the highest level with supervoxels having relatively larger spatial extents. Notice, the supervoxels belong to higher levels of segmentation hierarchy tend to preserve the semantic information and are less prone to leaks. Also note that, video segmentation is computed on the entire video clip, for the sake of visualization, we pick three sample frames.
model parameters. The final outputs of the human motion segmentation are the human foreground background binary maps as depicted in Figure 5.6.

![Figure 5.6](image)

**Figure 5.6:** (a) Three sample input video frames showing a “handshaking” action from a test video clip of LIRIS HARL dataset []. (b) The final human action foreground-background segmentation results are shown for these three frames.

**Proposal based on Selective Search.** We use two competing approaches to generate region proposals for action detection. The first is based upon Selective Search [37], and the second approach is presented in Section 5.3.1. Whilst using the Selective Search based method for both training and testing, we only use the motion segmentation based method for testing since it does not provide good negative proposals to use during training. Having a sufficient number of negative examples is crucial to train an effective classifier. At test time, the human motion segmentation (§ Section 5.3.1) allows us to extract pixel-level action instance segmentation which is superior to what we may obtain by using Selective Search. We validate our action detection pipeline using both algorithms - the results are discussed in Section 5.4.

**Measuring “actionness” of Selective Search proposals.** The selective-search region-merging similarity score is based on a combination of colour (histogram intersection), and size properties, encouraging smaller regions to merge
early, and avoid holes in the hierarchical grouping. Selective Search (SS) generates on average 2,000 region proposals per frame, most of which do not contain human activities. In order to rank the proposals with an “actionness” score and prune irrelevant regions, we compute dense optical flow between each pair of consecutive frames using the state-of-the-art algorithm in [47]. Unlike Gkioxari and Malik [2], we use a relatively smaller motion threshold value to prune SS boxes, (cf. Section A.2) to avoid neglecting human activities which exhibit minor body movements exhibited in the LIRIS HARL [14] such as “typing on keyboard”, “telephone conversation” and “discussion” activities. In addition to pruning region proposals, the 3-channel optical flow values (i.e., flow-\(x\), flow-\(y\) and the flow magnitude) are used to construct ‘motion images’ from which CNN motion features are extracted [2].

5.3.2 Appearance- and motion-based detection networks

In the second stage of the pipeline, we use the “actionness” ranked region proposals (cf. 5.3.1) to select image patches from both the RGB (original video frames) and flow images. The image patches are then fed to a pair of fine-tuned Convolutional Neural Networks (Figure 5.2(d)) (which encode appearance and local image motion, respectively) from which appearance and motion feature vectors were extracted. As a result the first network learns static appearance information (both lower-level features such as boundary lines, corners, edges and high level features such as object shapes), while the other encodes action dynamics at frame level. The output of the Convolutional Neural Network may be seen as a highly nonlinear transformation \(\Phi(\cdot)\) from local image patches to a high-dimensional vector space in which discrimination may be performed accurately even by a linear classifier. We follow the network architectures of [92] and [152].

**Pre-training.** We adopt a CNN training strategy similar to [1]. Indeed, for domain-specific tasks on relatively small scale datasets, such as LIRIS HARL [14], it is important to initialise the CNN weights using a model pre-trained on a larger-scale dataset, in order to avoid over-fitting [2]. Therefore, to encode object “context” we initialise the appearance-based CNN’s weights using a model pre-trained on the PASCAL VOC 2012’s object detection dataset. To encode typical motion patterns over a temporal window, the optical motion-based CNN is initialised using a model pre-trained on the UCF101 dataset (split 1) [8].
Chapter 5. Spatio-temporal Action Instance Segmentation and Localisation

Fine tuning. We use deep learning software tool Caffe \[153\] to fine-tune pre-trained domain-specific appearance- and motion-based CNNs on LIRIS HARL training set. For training CNNs, the Selective Search region proposals (§5.3.1) with an intersection-union overlap score greater than 0.5 with respect to the ground truth bounding box were considered as positive examples, the rest as negative examples. The image patches specified by the pruned region proposals were randomly cropped and horizontally flipped by the Caffe’s WindowDataLayer \[153\] with a crop dimension of $227 \times 227$ and a flip probability of 0.5 (Figure 5.2 (c)). Random cropping and flipping were done for both RGB and flow images. The pre-processed image patches along with the associated ground-truth action class labels are then passed as inputs to the appearance and motion CNNs to fine-tune (i.e. updating only the weights of the fully connected layers, in this case, fc6 and fc7 layers, and keeping the weights of the other layers untouched during training) for action classification (Figure 5.2 (d)). A mini batch of 128 image patches (32 positive and 96 negative examples) are processed by the CNNs at each training forward-pass. Note that the number of batches varies frame-to-frame as per the number of ranked proposals per frame. It makes sense to include fewer positive examples (action regions) as these are relatively rare when compared to background patches (negative examples).

[I have a technical note which have all details of fc7 feature normalisation, scaling, svm training algorithms etc. Put those in Appendix and refer them here accordingly.]

Feature extraction from CNN layers. We extract the appearance- and motion-based features from the fc7 (fully connected layer 7) layer of the two networks. Thus, we get two feature vectors (each of dimension 4096): appearance feature $x_a = \Phi_a(r)$ and motion feature $x_f = \Phi_f(r)$. We perform L2 normalisation on the obtained feature vectors, to then, scale and merge appearance and motion features (Figure 5.2 (d)) in an approach similar to that proposed by \[2\]. This yields a single feature vector $x$ for each image patch $r$. Such frame-level region feature vectors are used to train an SVM classifier (§ Section 5.3.3).

5.3.3 Training region proposal classifiers

Once discriminative CNN fc7 feature vectors $x \in \mathbb{R}^n$ are extracted for region proposals (§5.3.1), they can be used to train a set of binary classifiers (Figure 5.2 (d)) to attach a vector of scores $s_c$ to each region proposal $r$, where each element in the score vector $s_c$ is a confidence measure of each action class
\( c \in \{1, 2, \ldots, C\} \) to be present within that region. Due to the notable success of linear SVM classifiers when combined with CNN features [1], we trained a set of 1-vs-rest linear SVMs to classify region proposals.

**Class specific positive and negative examples.** In contrast to the RCNN-based one-vs-rest training approach of [1], in which only the the ground-truth bounding boxes are considered as positive examples, due to extremely high inter- and intra-class variations in LIRIS HARL dataset [14], we use as positive examples: the ground truth + those bounding boxes which have an overlap with the ground-truth greater than 75%, which we think is more intuitive for complex datasets to train SVMs with more positive examples rather than only ground-truth. We achieved almost 5% gain over SVMs classification accuracy with this training strategy. In a similar way, we consider as negative examples only those features vectors whose associated region proposal have an overlap smaller than 30% with respect to the ground truth bounding boxes (possibly several) present in the frame.

**Training with hard negative mining.** We train the set of class specific linear SVMs using hard negative mining [91] to speed up the training process. Namely, in each iteration of the SVM training step we consider only those negative features which fall within the margin of the decision boundary. We use the publicly available toolbox *Liblinear* [1] for SVM training and use \( L_2 \) regularizer and \( L_1 \) hinge-loss with the following parameter values to train the SVMs: positive loss weight \( W_{LP} = 2 \); SVM regularisation constant \( C = 10^{-3} \); bias multiplier \( B = 10 \).

### 5.3.4 Testing region proposal classifiers

With our actionness-ranked region proposals \( r_i \) (§5.3.1) we can extract a cropped image patch and pass it to the CNNs for feature extraction in a similar fashion as described in Section 5.3.2. A prediction takes the form:

\[
s_c(r) = w_c^T \Phi(r) + b_c, \tag{5.1}
\]

where, \( \Phi(r) = \{\Phi_a(r); \Phi_f(r)\} \) is combination of appearance and motion features of \( r \), \( w_c^T \) and \( b_c \) are the hyperplane parameter and the bias term of the learned SVM model of class \( c \). The confidence measure \( s_c(r) \) that the action ‘\( c \)’ has

happened in region ‘r’ is based on the appearance and motion features. Due to the typically large number of region proposals generated by the Selective Search algorithms (§5.3.1), we further apply non-maximum suppression to prune the regions.

5.3.5 Action tube generation and classification

Since our region proposals are generated on each video frame, linking these regions in space and time is essential to generate action tubes. We use our two-pass dynamic programming approach presented in Section 4.3.4 to formulate the action tube detection problem as a labelling problem which is divided into two parts: i) we link the spatial regions into temporally connected action paths for each action, and ii) we perform a piece-wise constant temporal labelling on the action paths.

Note for Suman: for clarity, I am going to put the action tube generation section of this chapter in Appendix section. Putting it in main chapter is redundant as most of the main ideas are already presented in bmvc2016 work.

5.4 Experimental results

We evaluate two region proposal methods with our pipeline, one based on human motion segmentation (HMS) (§5.3.1) and another one based on selective search (SS) (§5.3.1). We will use “HMS” and “SS” abbreviations in tables and plot to show the performance of our pipeline based on each region proposal technique. Our results are also compared to the current state-of-the-art: VPULABUAM-13 [143] and IACAS-51 [144].

5.4.1 Instance classification performance - no localisation (NL).

This evaluation strategy ignores the localisation information (i.e. the bounding boxes) and only focuses on whether an action is present in a video or not. If a video contains multiple actions then system should return the labels of all the actions present correctly. Even though our action detection framework is not specifically designed for this task, we still outperform the competition, as shown in Table 5.1.
5.4.2 Detection and localisation performance.

This evaluation strategy takes localisation (space and time) information into account\[145\]. We use a 10% threshold quality level for the four thresholds (§ 3.0.1), which is the same as that used in the LIRIS-HARL competition. In Table 5.1 we denote these results as “method-name-NL” (NL for no localisation) and “method-name-10%”. In both cases (without localisation and with 10% overlap), our method outperforms existing approaches, achieving an improvement from 46%\[143\] to 56%, in terms of F1 score without localisation measures, and a improvement from 5%\[143\] to 56% (11.2 times better) gain in the F1-score when 10% localisation information is taken into account. In Table 5.2 we list the results we obtained using the overall integrated performance scores (Equation 3.1) - our method yields significantly better quantitative and qualitative results with an improvement from 3%\[143\] to 43% (14.3% times better) in terms of F1 score, a relative gain across the spectrum of measures. Samples of qualitative instance segmentation results are shown in Fig. 5.7.

The pure classification accuracy of the HMS- and SS-based approaches are reflected in the Confusion Matrices shown in Figure 5.9. Confusion matrices

Figure 5.7: Correct (a-c) and incorrect (d-f) instance segmentation results on the LIRIS-HARL dataset, the correct category is shown in brackets. (a) ‘Try enter room unsuccessfully’. (b) ‘Discussion’. (c) ‘Unlock enter/leave room’. (d) ‘Handshaking’ (Give take object from person). (e) ‘Discussion’ (Leave bag unattended). (f) ‘Put take object into/from desk’ (Telephone conversation).
Table 5.1: Quantitative measures precision and recall.

<table>
<thead>
<tr>
<th>Method</th>
<th>Recall</th>
<th>Precision</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPULABUAM-13-NL</td>
<td>0.36</td>
<td>0.66</td>
<td>0.46</td>
</tr>
<tr>
<td>IACAS-51-NL</td>
<td>0.3</td>
<td>0.46</td>
<td>0.36</td>
</tr>
<tr>
<td>SS-NL (ours)</td>
<td>0.5</td>
<td>0.53</td>
<td>0.52</td>
</tr>
<tr>
<td>HMS-NL (ours)</td>
<td>0.5</td>
<td>0.63</td>
<td>0.56</td>
</tr>
<tr>
<td>VPULABUAM-13-10%</td>
<td>0.04</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td>IACAS-51-NL-10%</td>
<td>0.03</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>SS-10% (ours)</td>
<td>0.5</td>
<td>0.53</td>
<td>0.52</td>
</tr>
<tr>
<td>HMS-10% (ours)</td>
<td>0.5</td>
<td>0.63</td>
<td>0.56</td>
</tr>
</tbody>
</table>

Table 5.2: Qualitative thresholds and integrated score.

<table>
<thead>
<tr>
<th>Method</th>
<th>$I_{sr}$</th>
<th>$I_{sp}$</th>
<th>$I_{tr}$</th>
<th>$I_{tp}$</th>
<th>IQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPULABUAM-13-IQ</td>
<td>0.02</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>IACAS-51-IQ</td>
<td>0.01</td>
<td>0.01</td>
<td>0.03</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td>SS-IQ (ours)</td>
<td>0.52</td>
<td>0.22</td>
<td>0.41</td>
<td>0.39</td>
<td>0.38</td>
</tr>
<tr>
<td>HMS-IQ (ours)</td>
<td>0.49</td>
<td>0.35</td>
<td>0.46</td>
<td>0.43</td>
<td>0.44</td>
</tr>
</tbody>
</table>

show the complexity of dataset. Some of the actions are completely confusing with others, for eg. telephone-conversation is completely confused with put-take-object-to-from-box-desk, same can be observed for action unlock-enter-leave-room in SS approach.

5.4.3 Performance vs detection quality curves.

The plots in Figure 5.8 attest the robustness of our method, as they depict the curves corresponding to precision, recall and F1-score over varying quality thresholds.

When the threshold $t_{tr}$ for temporal recall is considered (§ Figure 5.8 plot-(a)) we achieved a highest recall of 50% for both HMS- and SS-based approaches and a highest precision of 65% for HMS-based approach at threshold value of $t_{tr}=0$. As the threshold increases towards $t_{tr} = 1$, SS-based method shows a robust performance, with highest recall=50% and precision=52%, HMS-based method shows promising results with an acceptable drop in precision and recall. Note that when $t_{tr}=1$, we assume that all frames of an activity instance need to be detected in order for the instance itself to be considered as detected. As for the competing methods, IACAS-51 [144] yields the next competing recall of 2.4% and a precision of 3.7% with a threshold value of $t_{tr}=1$. 
When acting on the value of the temporal frame-wise precision threshold $t_{tp}$ (§ Figure 5.8 plot-(b)) we can observe that at $t_{tp}=1$, when we assume that not a single spurious frame outside the ground-truth temporal window is allowed, our HMS-based region proposal approach gives highest recall of 8% and precision 10.7%, where, as SS-based approach has significantly lower recall=2% and precision=2.4%, which is still significantly higher than the performance of the existing methods. Indeed, at $t_{tp}=1$, VPULABUAM-13 has recall=0.8% and precision=1% where IACAS-51 yields both zero precision and zero recall. This results tell us that HMS-based approach performs superior in detecting temporal extent of an action and thus is suitable for action localisation in temporally
untrimmed videos. The remaining two plots-(c) and -(d) of Figure 5.8 illustrate the overall performance when spatial overlap is taken into account. Both plots show metrics approaching zero when the corresponding spatial thresholds (pixel-wise recall $t_{sr}$ and pixel-wise precision $t_{sp}$) approach 1. Note that it is highly unlikely for a ground truth activity to be consistently (spatially) included in the corresponding detected activity over all the consecutive frames (spatial recall), as indicated in the plot-(c). It is also rare for a detected activity to be (spatially) included in the corresponding ground-truth activity over all the frames (spatial precision) as indicated in plot-(d).

For the pixel-wise recall (plot-(c)), our HMS based method shows consistent recall between 45% to 50% and precision between 59% to 65.5% up to a threshold value of $t_{sr}=0.7$, whereas SS-based region proposal approach gives comparable recall between 48.3% to 50.8%, but relative lower precision between 43.5% to 53.2% up to $t_{sp}=0.7$. For the pixel-wise precision (plot-(d)), HMS and SS-based approaches give similar recall between 39% to 50%, whereas HMS-method again outperforms in precision with 48% to 63% up to a threshold value of $t_{sp}=0.7$, whereas SS has precision 41% to 53% up to a threshold value $t_{sp}=0.7$. Finally,
we draw conclusion that our HMS-based region proposal approach shows superior qualitative and quantitative detection performance on the challenging LIRIS HARL dataset.

5.4.4 Additional qualitative detection results.

Figure 5.10 shows additional qualitative action instance segmentation and localisation results on LIRIS HARL dataset.

Figure 5.10: Additional qualitative action instance segmentation and localisation results on LIRIS HARL dataset
Chapter 6

AMTnet: Action-Micro-Tube Regression Using Deep Architecture

[Fabio, please give a pass to this intro section.]

6.1 Introduction

In previous chapters (Chapter 4 and Chapter 5), we present action detection frameworks which are based on frame-level representation (cf. Section 1.5.1) resulting a suboptimal solutions to the action detection problem (cf. Section 1.5.2). In this chapter, we propose a new deep network architecture which exploits the video-level representation and 3D region proposals (cf. Section 1.5.3) to classify and regress whole video subsets.

Thus, we take a first step towards a truly optimal solution of the action detection problem by considering video-level space-time action region hypotheses formed by a pair of bounding boxes spanning two successive video frames at an arbitrary temporal interval $\Delta$ (cf. Section 1.5.3, Figure 1.10 & Figure 6.2). We call these pairs of bounding boxes 3D region proposals. The advantages of this approach are that a) appearance features can be exploited to learn temporal dependencies (unlike what happens in current approaches), thus boosting detection performance; b) the linking of frame-level detections over time (i.e. solving the inter-frame data association problem (cf. Section 1.3)) is no longer a post processing step and can be (partially) learned by the network. Obviously, at this stage we still need to construct action tubes from 3D region proposals.

We thus propose a radically new approach to action detection based on (1)
Figure 6.1: At train time, the input to the network is a pair of successive video frames (a) which are processed through two parallel VGG-16 networks (b). The feature maps generated by the last convolution layers are fused (c) and the fused feature map is fed to a 3D-RPN network (d). The RPN generates 3D region proposals and their associated actionness scores which are then sampled as positive and negative training examples (f) by a proposal sampler (e). The sampled proposals and their scores are used to compute the actionness and 3D proposal regression losses (g). Subsequently, a bilinear feature pooling (h) and an element-wise feature fusion (i) are used to obtain a fixed sized feature representation for each sampled 3D proposal. Finally, the pooled and fused features are passed through fully connected (FC6 & FC7) classification and regression layers to train for action classification and a micro-tube regression. At test time, the predicted micro-tubes are linked in time by the action-tube generator (m).

(a) Video frames $f_t$ 
(b) VGG-16 network 
(c) Feature map fusion 
(d) 3D-RPN network 
(e) Region proposal sampler 
(f) Positive & negative samples 
(g) RPN loss computation 
(h) Bilinear interpolation 
(i) Proposal feature fusion 
(j) FC6 and FC7 layers 
(k) Actionness & Reg losses 
(l) Cls & Reg layers 
(m) Action-tube generation

Figure 6.2: (a) The 3D region proposals generated by our 3D-RPN network span pairs of successive video frames $f_t$ and $f_{t+\Delta}$ at temporal distance $\Delta$. (b) Ground-truth action-micro-tubes generated from different pairs of successive video frames.

A novel deep learning architecture for classifying space-time action regions and regressing 3D proposals (cf. Section 1.5.3 & Figure 1.10) illustrated in Figure 6.1 in combination with (2) an original strategy for linking micro-tubes up into proper action tubes. At test time, this new framework does not completely rely on post-processing for assembling frame-level detections, but makes use of the temporal encoding learned by the network. We show that: i) such a network trained on pairs of successive RGB video frames can learn the spatial
and temporal extents of action instances relatively better than those trained on individual video frames, and ii) our model outperforms the current state-of-the-art \cite{2,3,6} in spatio-temporal action detection by just exploiting appearance (the RGB video frames), in opposition to the methods which heavily exploit expensive optical flow maps.

Just to be clear, the aim of this chapter is not to renounce to optical flow cues, but to move from frame-level detections to whole tube regression. Indeed the method can be easily extended to incorporate motion at the micro-tube level rather than frame level, allowing fusion of appearance and motion at training time, unlike current methods \cite{9,11}. An extension of this work which incorporates optical flow signals is presented in Chapter \cite{7}.

Outline. coming soon ...

Related publication. The work presented in this chapter has appeared in ICCV 2017 \cite{10}. In Section 6.7 and 6.8, we refer the work presented in Chapter \cite{4} with its related BMVC 2016 publication \cite{6}.

6.2 Overview of the approach

Our proposed network architecture (see Figure \ref{fig:6.1}) employs and adapts some of the architectural components recently proposed in \cite{7,51}. At training time, the input to the model is a pair of successive video frames (a) which are fed to two parallel CNNs (b) (§ Section 6.3.1). The output feature maps of the two CNNs are fused (c) and passed as input to a 3D region proposal network (3D-RPN) (d) (§ Section 6.3.2). The 3D-RPN network generates 3D region proposals and their associated actionness\footnote{The term actionness \cite{39} is used to denote the possibility of an action being present within a 3D region proposal.} scores, which are then sampled as positive and negative training examples (f) by a proposal sampler (e) (§ 6.3.3). A training mini-batch of 256 examples are constructed from these positive and negative samples. The mini-batch is firstly used to compute the actionness classification and 3D proposal regression losses (g) (§ 6.4.1), and secondly, to pool CNN features (for each 3D proposal) using a bilinear interpolation layer (h) (§ 6.3.4).

In order to interface with the fully connected layers (j) (§ 6.3.5), bilinear interpolation is used to get a fixed-size feature representation for each variably sized 3D region proposal. As our 3D proposals consist of a pair of bounding
boxes, we apply bilinear feature pooling independently on each bounding box in a pair, which gives rise to two fixed-size pooled feature maps of size $[512 \times kh \times kw]$, where $kh = kw = 7$ for each 3D proposal. We then apply element-wise fusion (§6.3.4) to these 2 feature maps. Each pooled and then fused feature map (representing a 3D proposal) is passed to two fully connected layers (FC6 and FC7) (§6.3.5). The output of the FC7 layer is a fixed sized feature vector of shape $[4096 \times 1]$. These 4096 dimension feature vectors are then used by a classification and a regression layers (k) (§6.3.5) to output (1) $B \times C$ classification scores and (2) $B \times C \times 8$ coordinate values where $B$ is the number of 3D proposals in a training mini-batch and $C$ is the number of action categories in a given dataset.

At test time we select top 1000 predicted micro-tubes by using non-maximum suppression, modified to work with pairs of bounding boxes and pass these to an action-tube generator (m) (§6.5) which links those micro-tubes in time. At both training and test time, our model receives as input successive video frames $f_t, f_{t+\Delta}$. At training time we generate training pairs using 2 different $\Delta$ values 1 and 2 (§6.6.1). At test time we fix $\Delta = 1$. As we show in the Section 6.8.6 even consecutive frames ($\Delta = 1$) carry significantly different information which affects the overall video-mAP. Throughout this chapter, “3D region proposals” denotes the RPN-generated pairs of anchor boxes regressed by the middle layer (Figure 6.1 (g)), and later by the end layer (Figure 6.1 (l)). Whereas, “micro-tubes” refers to the pairs of either ground-truth bounding-boxes or pairs of detection bounding-boxes predicted by the network at test time (cf. Section 1.5.3 & Figure 1.10).

### 6.3 Network Architecture

All the stages of Figure 6.1 are described below in detail.

![Figure 6.3: 3D-RPN architecture.](image)
6.3.1 Convolutional Neural Network

The convolutional (conv) layers of our network follow the VGG-16 architecture \cite{Simonyan2014}. We use two parallel VGG-16 networks (§ Figure 6.1 (b)) to apply convolution over a pair of successive video frames. Each VGG-16 has 13 conv layers intermixed with 5 max pooling layers. Each conv layer has a $3 \times 3$ filter and $1 \times 1$ stride and padding. Each max pooling layer has filter shape $2 \times 2$. We discard all the VGG-16 layers after the last (13-th) conv layer.

**Feature map fusion.** Our network takes two successive video frames $f_t$ and $f_{t+\Delta}$ as inputs. For a input video frame of shape $[3 \times H \times W]$, the last conv layer of each VGG-16 outputs a feature map of shape $[D \times H' \times W']$ where $D = 512$, $H' = \frac{H}{16}$, and $W' = \frac{W}{16}$. We fuse the two conv feature maps produced by the two parallel VGG-16 networks using element-wise sum fusion (§ Figure 6.1 (c)).

As a consequence, the fused feature map encodes both appearance and motion information (for frames $f_t$ and $f_{t+\Delta}$), which we pass as input to our 3D-RPN network.

Our new 3D region proposal network (Figure 6.1 (d)) builds on the basic RPN structure \cite{Ren2015} to propose a fully convolutional network which can generate 3D region proposals via a number of significant architectural changes.

6.3.2 3D region proposal network

**3D region proposal generation.** As we explained, unlike a classical RPN \cite{Ren2015} which generates region proposals (rectangular bounding boxes) per image, our 3D-RPN network generates (video) region proposals spanning a pair of video frames. A single proposal thus consists of a pair of rectangular bounding boxes. The input to our 3D-RPN is a fused VGG-16 feature map (§ Figure 6.1 (c)) of size $[512 \times H' \times W']$. We generate anchor boxes in a similar way as in \cite{Ren2015}: namely, we project back each point in the $H' \times W'$ grid (of the input feature map) onto the original image plane of size $H \times W$. For each projected point we generate $k$ pairs of anchor boxes of different aspect ratios.

Let $(x_{a_i}, y_{a_i}, w_{a_i}, h_{a_i})$ denote the centroid, width and height of the anchor boxes in a pair. We use the subscript $i$ to index the two boxes in a pair, i.e. $i = \{1, 2\}$. Similarly, $(x_{g_i}, y_{g_i}, w_{g_i}, h_{g_i})$ refer to the centroid, width and height of the ground-truth pair. We can transform a pair of input anchor boxes into a predicted pair
of ground-truth boxes via\footnote{We removed the subscript \(i\) in Eq. \ref{eq:6.1} for sake of simplicity.}:

\[
\begin{align*}
    x_g &= x_a + \phi_x w_a \\
    y_g &= y_a + \phi_y h_a \\
    w_g &= w_a \exp(\phi_w) \\
    h_g &= h_a \exp(\phi_h)
\end{align*}
\] (6.1)

where \((\phi_x, \phi_y)\) specify a scale-invariant translation of the center of the anchor boxes, and \((\phi_w, \phi_h)\) specify a log-space translation of their width and height.

Both RPN and the micro-tube regression layer (Figure \ref{fig:6.1}(k)) predict the bounding box regression offsets \((\phi_x, \phi_y, \phi_w, \phi_h)\). Our anchor generation approach differs from that of \cite{7}, in the sense that we generate \(k\) pairs of anchors instead of \(k\) anchors.

**Network architecture.** The network architecture of our 3D-RPN is depicted in Figure \ref{fig:6.3}. To encode the location information of each pair of anchors, we pass the fused VGG-16 feature map through a \(3 \times 3\) convolution (b), a rectified linear nonlinearity (c), and two more \(1 \times 1\) convolution ((e) and (h)) layers. The first conv layer (b) consists of 256 convolution filters with \(1 \times 1\) stride and padding, resulting in a feature map of size \([256 \times H' \times W']\) (d). The second conv layer (e) has \(8 \times k\) convolution filters with \(1 \times 1\) stride and does not have padding. It outputs a feature map of shape \([(8 \times k) \times H' \times W']\) (f) which encodes the location information (8 coordinate values) of \([k \times H' \times W']\) pairs of anchor boxes (g). The third conv layer (h) is the same as (e). The only difference is in the number of filters which is \(2 \times k\) to encode the actionness score (i.e. probability of action or no-action) (j) for each \(k\) pairs of anchors.

As RPN is a fully convolutional neural network, classification and regression weights are learned directly from the convolution features, whereas in the fully connected layers (§6.3.5) we apply linear transformation layers for classification and regression. In our 3D-RPN, the convolution layer (e) is considered as the regression layer, as it outputs the 8 regression offsets per pair of anchor boxes; the convolution layer (h) is the classification layer.

### 6.3.3 3D region proposal sampling

Processing all the resulting region proposals is very expensive. For example, with \(k = 12\) and a feature map of size \([512 \times 38 \times 50]\), we get \(12 \times 38 \times 50 = 22800\) pairs of anchor boxes. For this reason, we subsample them during both training and testing following the approach of \cite{7} (§ Figure \ref{fig:6.1}(e)). We only make a
slight modification in the sampling technique, as in our case one sample consists of a pair of bounding boxes, rather than a single box.

**Training time sampling.** During training, we compute the intersection over union (IoU) between a pair of ground-truth boxes \( \{G_t, G_{t+\Delta}\} \) and a pair of proposal boxes \( \{P_1, P_2\} \), so that, \( \psi_1 = \text{IoU}(G_t, P_1) \) and \( \psi_2 = \text{IoU}(G_{t+\Delta}, P_2) \). We consider \( \{P_1, P_2\} \) as a positive example if \( \psi_1 > 0.5 \) and \( \psi_2 > 0.5 \), that is both IoU values are above 0.5. When enforcing this condition, there might be cases in which we do not have any positive pairs. To avoid such cases, we also consider as positive pairs those which have maximal mean IoU \( \frac{\psi_1 + \psi_2}{2} \) with the ground-truth pair. As negative examples we consider pairs for which both IoU values are below 0.3.

We construct a minibatch of size \( B \) in which we can have at most \( B_p = B/2 \) positive and \( B_N = B - B_p \) negative training samples. We set \( B = 256 \). Note that the ground-truth boxes \( \{G_t, G_{t+\Delta}\} \) in a pair belong to a same action instance but come from two different video frames \( \{f_t, f_{t+\Delta}\} \). As there may be multiple action instances present, during sampling one needs to make sure that a pair of ground-truth boxes belongs to the same instance. To this purpose, we use the ground-truth tube-id provided in the datasets to keep track of instances.

**Test time sampling.** During testing, we use non-maximum suppression (NMS) to select the top \( B = 1000 \) proposal pairs. We made changes to the NMS algorithm to select the top \( B \) pairs of boxes based on their confidence. In NMS, one first selects the box with the highest confidence, to then compute the IoU between the selected box and the rest. In our modified version (i) we first select the pair of detection boxes with the highest confidence; (ii) we then compute the mean IoU between the selected pair and the remaining pairs, and finally (iii) remove from the detection list pairs whose IoU is above an overlap threshold \( th_{nms} \).

### 6.3.4 Bilinear Interpolation

The sampled 3D region proposals are of different sizes and aspect ratios. We use bilinear interpolation \[49, 50\] to provide a fixed-size feature representation for them, necessary to pass the feature map of each 3D region proposal to the fully connected layer fc6 of VGG-16 (§ Figure 6.1(j)), which indeed requires a fixed-size feature map as input. Whereas recent action detection methods \[6, 11\] use max-pooling of region of interest (RoI) features which only backpropagates
the gradients w.r.t. convolutional features, bilinear interpolation allows us to backpropagate gradients with respect to both (a) convolutional features and (b) 3D RoI coordinates. Further, whereas \[6, 11\] train appearance and motion streams independently, and perform fusion at test time, our model requires one-time training, and feature fusion is done at training time.

**Feature fusion of 3D region proposals.** As a 3D proposal consists of a pair of bounding boxes, we apply bilinear feature pooling independently to each bounding box in the pair. This yields two fixed-size pooled feature maps of size \(D \times kh \times kw\) for each 3D proposal. We then apply element-wise sum fusion (§Figure 6.1 (i)) to these 2 feature maps, producing an output feature map of size \(D \times kh \times kw\). Each fused feature map encodes the appearance and motion information of (the portion of) an action instance which may be present within the corresponding 3D region proposal. In this work, we use \(D = 512, kh = kw = 7\).

### 6.3.5 Fully connected layers

Our network employs two fully connected layers FC6 and FC7 (Figure 6.1 (j)), followed by an action classification layer and a micro-tube regression layer (Figure 6.1 (k)).

The fused feature maps (§Section 6.3.4) for each 3D proposal are flattened into a vector and passed through FC6 and FC7. Both layers use rectified linear units and dropout regularisation [51]. For each 3D region proposal, the FC7 layer outputs a 4096 dimension feature vector which encodes the appearance and motion features associated with the pair of bounding boxes. Finally, these 4096-dimensional feature vectors are passed to the classification and regression layers. The latter output \(B \times C\) softmax scores and \(B \times C \times 8\) bounding box regression offsets (§6.3.2), respectively, for \(B\) predicted micro-tubes and \(C\) action classes.

### 6.4 Network training

#### 6.4.1 Multi-task loss function

As can be observed in Figures 6.1 and 6.3, our network contains two distinct classification layers. The mid classification layer (§Figure 6.3 (h)) predicts the probability \(p^m\) of a 3D proposal containing an action, \(p^m = (p^m_0, p^m_1)\) over two
classes (action vs. no action). We denote the associated loss by $L_{cls}^m$. The end classification layer (§ Figure 6.1 (k)) outputs a discrete probability distribution (per 3D proposal), $p^e = (p^e_0, ..., p^e_C)$, over $C + 1$ action categories. We denote the associated loss by $L_{cls}^e$.

In the same way, the network has a mid (Figure 6.3 (e)) and an end (§ Figure 6.1 (k)) regression layer – the associated losses are denoted by $L_{loc}^m$ and $L_{loc}^e$, respectively. Both regression layers output a pair of bounding box offsets $\phi^m$ and $\phi^e$ (cfr. Eq. 6.1). We adopt the parameterization of $\phi$ (§ 6.3.2) given in [1].

Now, each training 3D proposal is labelled with a ground-truth action class $c^e$ and a ground-truth micro-tube (cf. Section 1.5.3, Figure 1.10 & Figure 6.2) regression target $g^e$. We can then use the multi-task loss [7]:

$$L(p^e, c^e, \phi^e, g^e, p^m, c^m, \phi^m, g^m) =$$

$$\lambda_{cls}^e L_{cls}(p^e, c^e) + \lambda_{loc}^e [c \geq 1]L_{loc}(\phi^e, g^e) +$$

$$\lambda_{cls}^m L_{cls}(p^m, c^m) + \lambda_{loc}^m [c = 1]L_{loc}(\phi^m, g^m)$$

(6.2)

on each labelled 3D proposal to jointly train for (i) action classification ($p^e$), (ii) micro-tube regression ($\phi^e$), (iii) actionness classification ($p^m$), and (iv) 3D proposal regression ($\phi^m$). Here, $L_{cls}^e(p^e, c^e)$ and $L_{cls}^m(p^m, c^m)$ are the cross-entropy losses for the true classes $c^e$ and $c^m$ respectively, where $c^m$ is 1 if the 3D proposal is positive and 0 if it is negative, and $c^e = \{1, ..., C\}$.

The second term $L_{loc}^e(\phi^e, g^e)$ is defined over an 8-dim tuple of ground-truth micro-tube regression target coordinates:

$$g^e = \left(\{g^e_{x_1}, g^e_{y_1}, g^e_{w_1}, g^e_{h_1}\}, \{g^e_{x_2}, g^e_{y_2}, g^e_{w_2}, g^e_{h_2}\}\right)$$

and the corresponding predicted micro-tube tuple:

$$\phi^e = \left(\{\phi^e_{x_1}, \phi^e_{y_1}, \phi^e_{w_1}, \phi^e_{h_1}\}, \{\phi^e_{x_2}, \phi^e_{y_2}, \phi^e_{w_2}, \phi^e_{h_2}\}\right).$$

The fourth term $L_{loc}^m(\phi^m, g^m)$ is similarly defined over a tuple $g^m$ of ground-truth 3D proposal regression target coordinates and the associated predicted tuple $\phi^m$. The Iverson bracket indicator function $[c \geq 1]$ in (6.2) returns 1 when $c^e \geq 1$ and 0 otherwise; $[c = 1]$ returns 1 when $c^m = 1$ and 0 otherwise.

For both regression layers we use a smooth $L1$ loss in transformed coordinate space as suggested by [7]. The hyper-parameters $\lambda_{cls}^e$, $\lambda_{loc}^e$, $\lambda_{cls}^m$ and $\lambda_{loc}^m$, in Eq. 6.2
weigh the relative importance of the four loss terms. In the following we set to 1 all four hyper-parameters.

### 6.4.2 Optimisation

We follow the end-to-end training strategy of [51] to train the entire network in a single optimisation step. We use stochastic gradient descent (SGD) to update the weights of the two VGG-16 convolutional networks, with a momentum of 0.9. To update the weights of other layers of the network, we use the Adam [154] optimiser, with parameter values $\beta_1 = 0.9$, $\beta_2 = 0.99$ and a learning rate of $1 \times 10^{-6}$. During the 1st training epoch, we freeze the weights of the convolution networks and update only the weights of the rest of the network. We start fine-tuning the layers of the two parallel CNNs after completion of 1st epoch. The first four layers of both CNNs are not fine-tuned for sake of efficiency. The VGG-16 pretrained ImageNet weights are used to initialise the convolutional nets. The rest of the network’s weights are initialised using a Gaussian with $\sigma = 0.01$.

### 6.5 Action-tube generation

![Diagram of action-tube generation](image)

Figure 6.4: (a) The temporal associations learned by our network; (b) Our micro-tube linking algorithm requires $(T/2 - 1)$ connections; (c) the $T - 1$ connections required by [6]’s approach.

Once the predicted micro-tubes are regressed at test time, they need to be linked up to create complete action tubes associated with an action instance. To do this we introduce here a new action tube generation algorithm which is an evolution of that presented in [6]. There, temporally untrimmed action paths are first generated in a first pass of dynamic programming. In a second pass, paths are temporally trimmed to detect their start and end time. Here we modify the
first pass of [6] and build action paths using the temporal associations learned by our network. We use the second pass without any modification.

Linking up micro tubes (§ Figure 6.4) is not the same as linking up frame-level detections as in [6]. In the Viterbi forward pass of [6], the edge scores between bounding boxes belonging to consecutive video frames (i.e., frame $f_t$ and $f_{t+1}$) are first computed. Subsequently, a DP (dynamic programming) matrix is constructed to keep track of the box indices with maximum edge scores. In the Viterbi backward pass, all consecutive pairs of frames, i.e., frames $\{1, 2\}, \{2, 3\}, \ldots$ are traversed to join detections in time. Our linking algorithm saves 50% of the computing time, by generating edge scores between micro-tubes (which only needs $T/2 - 1$ iterations, cfr. Figure 6.4) rather than between boxes from consecutive frames (which, in the forward pass, needs $T - 1$ iterations). In the backward pass, the algorithm connects the micro-tubes as per the max edge scores.

Recall that a predicted micro-tube consists of a pair of bounding boxes (§ Figure 6.4), so that $m = \{b^1, b^2\}$. In the first pass action-specific paths $p_c = \{m_t, t \in I = \{2, 4, ..., T - 2\}\}$, spanning the entire video length are obtained by maximising via dynamic programming [2]:

$$E(p_c) = \sum_{t \in I} s_c(m_t) + \lambda_o \sum_{t \in I} \psi_o(b_{m_t}^2, b_{m_t+2}^1),$$  

(6.3)

where $s_c(m_t)$ denotes the softmax score (§ 6.3.5) of the predicted micro-tube $m$ at time step $t$, the overlap potential $\psi_o(b_{m_t}^2, b_{m_t+2}^1)$ is the IoU between the second detection box $b_{m_t}^2$ which forms micro-tube $m_t$ and the first detection box $b_{m_t+2}^1$ of micro-tube $m_{t+2}$. Finally, $\lambda_o$ is a scalar parameter weighting the relative importance of the pairwise term. By recursively removing the detection micro-tubes associated with the current optimal path and maximising (6.3) for the remaining micro-tubes we can account for multiple co-occurring instances of the same action class.

### 6.6 Experimental setting

### 6.6.1 Evaluation metrics

As evaluation metrics we use both: (1) frame-AP (the average precision of detections at the frame level) as in [2, 11]; (2) video-AP (the average precision of detection at video level) as in [2, 3, 6, 11]. We select an IoU threshold ($\delta$)
range [0.1:0.1:0.5] for J-HMDB-21 and [0.1,0.2,0.3] for UCF-101 when computing video-mAP. For frame-mAP evaluation we set $\delta = 0.5$.

### 6.6.2 Training data sampling strategy

As the input to our model is a pair of successive video frames and their associated ground-truth micro-tubes, training data needs to be passed in a different way than in the frame-level training approach \cite{2,3,6,11}, where inputs are individual video frames. In our experiments, we use 3 different sampling schemes to construct training examples using different combinations of successive video frames (§ Figure 6.2 (b)): (1) scheme-11 generates training examples from the pairs of frames \{t=1,t=2\}, \{t=2,t=3\}, ...; scheme-21 uses the (non-overlapping) pairs \{1,2\}, \{3,4\}, ...; scheme-32 constructs training samples from the pairs \{1,3\}, \{4,6\}, ...

### 6.7 Model evaluation

We first show how a proper positive IoU threshold is essential during the sampling of 3D region proposals at training time (§ 6.3.3). Secondly, we assess whether our proposed network architecture, coupled with the new data sampling strategies (Sec. 6.6.1), improves detection performance. We then show that our model outperforms the appearance-based model of \cite{6}. Finally, we compare the performance of the overall detection framework with the state-of-the-art.

#### 6.7.1 Impact of different positive IoU thresholds on detection performance

We train our model on UCF-101 using two positive IoU thresholds: 0.7 and 0.5 (§ 6.3.3). The detection results (video-mAP) of these two models (Model-0.7 & -0.5) are shown in Table 6.1. Whereas \cite{7} recommends an IoU threshold of 0.7 to subsample positive region proposals during training, in our case we observe that an IoU threshold of 0.5 works better with our model. Indeed, during sampling we compute IoUs between pairs of bounding boxes and then take the mean IoU to subsample (§ 6.3.3). As the ground-truth boxes (micro-tubes) are connected in time and span different frames, it is harder to get enough positive examples with a higher threshold like 0.7. Therefore, in the remainder we use an IoU of 0.5 for evaluation.
Table 6.1: Impact of different positive IoU thresholds on detection performance (video-mAP).

<table>
<thead>
<tr>
<th>IoU threshold $\delta$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-0-7</td>
<td>64.04</td>
<td>54.83</td>
<td>44.664</td>
</tr>
<tr>
<td>Model-0-5</td>
<td>68.85</td>
<td>60.06</td>
<td>49.78</td>
</tr>
</tbody>
</table>

6.7.2 Impact of our training data sampling strategy on detection performance

JHMDB-21 frame-mAP. We first generate a J-HMDB-21 training set using the scheme-11 (§6.6.1) and train our model. We then generate another training set using scheme-32, and train our model on the combined training set (set-11+32). Table 6.3 and Table 6.3 show per class frame-APs and video-APs obtained using these two models. We can observe that out of 21 JHMDB action classes, the frame-APs of 15 classes and video-APs of 12 classes actually improve when training the model on the new combined trainset (set-11+32). Overall performance increases by 1.64% for frame-AP and 1.04% for video-AP at $\delta = 0.5$ indicating that the network learns temporal association more efficiently when it is trained on pairs generated from different combinations of successive video frames.

JHMDB-21 video-mAP. The two above trained models are denoted by Model-11 and Model-11+32 in Table 6.5 where the video-mAPs at different IoU threshold for these two models are shown. Although the first training strategy scheme-11 already makes use of all the video frames present in J-HMDB-21 training splits, when training our model using the combined trainset we observe an improvement in the video-mAP of 1.04% at $\delta = 0.5$.

6.7.3 Impact of exploiting appearance features

Further, we show that our model exploits appearance features (raw RGB frames) efficiently, contributing to an improvement of video-mAP by 3.2% over [6]. We generate a training set for UCF-101 split 1 using the training scheme-21 and compare our model’s performance with that of the appearance-based model (*A) of [6]. We show the comparison in Table 6.4.

Note that, among the 24 UCF-101 action classes, our model exhibits better video-APs for 14 classes, with an overall gain of 3.2%. We can observe that, al-
Table 6.2: Impact of our training data sampling strategy on per class frame-AP at IoU threshold $\delta = 0.5$, JHMDB-21 dataset (averaged over 3 splits).

<table>
<thead>
<tr>
<th>frame-AP(%)</th>
<th>brushHair</th>
<th>catch</th>
<th>clap</th>
<th>climbStairs</th>
<th>golf</th>
<th>jump</th>
<th>kickBall</th>
<th>pick</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours (*)</td>
<td>46.4</td>
<td>40.7</td>
<td>31.9</td>
<td>62.3</td>
<td>91.0</td>
<td>4.3</td>
<td>17.3</td>
<td>29.5</td>
</tr>
<tr>
<td>ours (**)</td>
<td>43.7</td>
<td>43.6</td>
<td>33.0</td>
<td>61.5</td>
<td>91.8</td>
<td>5.6</td>
<td>23.8</td>
<td>31.5</td>
</tr>
<tr>
<td>Improvement</td>
<td>-2.6</td>
<td>2.9</td>
<td>1.0</td>
<td>-0.8</td>
<td>0.7</td>
<td>1.2</td>
<td>6.4</td>
<td>1.9</td>
</tr>
</tbody>
</table>

Gkioxari et al. [2] 65.2 18.3 38.1 39.0 79.4 7.3 9.4 25.2
Wang et al. [134] 60.1 34.2 56.4 38.9 83.1 10.8 24.5 38.5
Weinzaepfel et al. [3] 73.3 34.0 40.8 56.8 93.9 5.9 13.8 38.5
Peng et al. [11] 75.8 38.4 62.2 62.4 99.6 12.7 35.1 57.8

<table>
<thead>
<tr>
<th>frame-AP(%)</th>
<th>pour</th>
<th>pullup</th>
<th>push</th>
<th>run</th>
<th>sBall*</th>
<th>sBow*</th>
<th>sGun*</th>
<th>sit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours (*)</td>
<td>86.2</td>
<td>82.7</td>
<td>66.9</td>
<td>35.5</td>
<td>33.9</td>
<td>78.2</td>
<td>49.7</td>
<td>11.7</td>
</tr>
<tr>
<td>ours (**)</td>
<td>91.8</td>
<td>84.1</td>
<td>73.1</td>
<td>32.3</td>
<td>33.3</td>
<td>81.4</td>
<td>55.1</td>
<td>12.4</td>
</tr>
<tr>
<td>Improvement</td>
<td>5.5</td>
<td>1.4</td>
<td>6.1</td>
<td>-3.2</td>
<td>-0.6</td>
<td>3.2</td>
<td>5.4</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Gkioxari et al. [2] 80.2 82.8 33.6 11.6 5.6 66.8 27.0 32.1
Wang et al. [134] 71.5 67.5 21.3 19.8 11.6 78.0 50.6 10.9
Weinzaepfel et al. [3] 88.1 89.4 60.5 21.1 23.9 85.6 37.8 34.9
Peng et al. [11] 96.8 97.3 79.6 38.1 52.8 90.8 62.7 33.6

<table>
<thead>
<tr>
<th>frame-AP(%)</th>
<th>stand</th>
<th>sBBall*</th>
<th>throw</th>
<th>walk</th>
<th>wave</th>
<th>mAP</th>
<th>–</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours (*)</td>
<td>13.8</td>
<td>57.1</td>
<td>21.3</td>
<td>27.8</td>
<td>27.1</td>
<td>43.6</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ours (**)</td>
<td>14.7</td>
<td>56.3</td>
<td>22.2</td>
<td>24.7</td>
<td>29.4</td>
<td>45.0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Improvement</td>
<td>0.8</td>
<td>-0.8</td>
<td>0.8</td>
<td>-3.1</td>
<td>2.3</td>
<td>1.4</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Gkioxari et al. [2] 34.2 33.6 15.5 34.0 21.9 36.2  – | –|
Wang et al. [134] 43.0 48.9 26.5 25.2 15.8 39.9  – | –|
Weinzaepfel et al. [3] 49.2 36.7 16.8 40.5 20.5 45.8  – | –|
Peng et al. [11] 48.9 62.2 25.6 59.7 37.1 58.5  – | –|

{Gkioxari [2], Wang [134], Weinzaepfel [3], Peng [11]} et al. exploit both appearance and flow features, whereas, our model only exploit appearance features.

sBall* : shootBall, sBow* : shootBow, sGun* : shootGun, sBBall* : swingBaseball.
*Model-11, **Model-11+32.

though trained on appearance features only, our model improves the video-APs significantly for action classes which exhibit a large variability in appearance and motion. Also, our model achieves relatively better spatiotemporal detection on action classes associated with video sequences which are significantly temporally untrimmed, such as BasketballDunk, GolfSwing, Diving with relative video-AP improvements of 16.9%, 10.8% and 1.5% respectively. We report significant gains in absolute video-AP for action categories SoccerJuggling, PoleVault, RopeClimbing, BasketballDunk, IceDancing, GolfSwing and LongJump of 45.6%, 22.1%, 19.4%, 16.9%, 11.2% 10.8% and 8.3%, respectively.
Table 6.3: Impact of our training data sampling strategy on per class video-AP at IoU threshold $\delta = 0.5$, JHMDB-21 dataset (averaged over 3 splits).

<table>
<thead>
<tr>
<th>video-AP(%)</th>
<th>brushHair</th>
<th>catch</th>
<th>clap</th>
<th>climbStairs</th>
<th>golf</th>
<th>jump</th>
<th>kickBall</th>
<th>pick</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours (*)</td>
<td>53.9</td>
<td>54.4</td>
<td>39.8</td>
<td>68.2</td>
<td>96.1</td>
<td>5.69</td>
<td>39.6</td>
<td>34.9</td>
</tr>
<tr>
<td>ours (**)</td>
<td>51.9</td>
<td>54.5</td>
<td>41.2</td>
<td>66.6</td>
<td>94.8</td>
<td>7.8</td>
<td>48.7</td>
<td>33.7</td>
</tr>
<tr>
<td>Improvement</td>
<td>-1.9</td>
<td><strong>0.01</strong></td>
<td><strong>1.4</strong></td>
<td>-1.6</td>
<td>-1.2</td>
<td><strong>2.1</strong></td>
<td><strong>9.1</strong></td>
<td>-1.2</td>
</tr>
<tr>
<td>Gkioxari et al. [2]</td>
<td>79.1</td>
<td>33.4</td>
<td>53.9</td>
<td>60.3</td>
<td>99.3</td>
<td>18.4</td>
<td>26.2</td>
<td>42.0</td>
</tr>
<tr>
<td>Wang et al. [134]</td>
<td>76.4</td>
<td>49.7</td>
<td>80.3</td>
<td>43.0</td>
<td>92.5</td>
<td>24.2</td>
<td>57.7</td>
<td>70.5</td>
</tr>
</tbody>
</table>

Gkioxari et al. [2], Wang et al. [134] et al. exploit both appearance and flow features, whereas, our model only exploit appearance features.

sBall* : shootBall, sBow* : shootBow, sGun* : shootGun, sBBall* : swingBaseball.

*Model-11, **Model-11+32.

Table 6.4: Per class video-AP comparison at IoU threshold $\delta = 0.2$, UCF-101-24 dataset.

<table>
<thead>
<tr>
<th>video-AP(%)</th>
<th>BaDu</th>
<th>Bi</th>
<th>Di</th>
<th>Fe</th>
<th>FlGy</th>
<th>GoSw</th>
<th>IcDa</th>
<th>LoJu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saha et al. [6] (*A)</td>
<td>22.7</td>
<td>56.1</td>
<td>89.7</td>
<td>86.9</td>
<td>93.8</td>
<td>59.9</td>
<td>59.2</td>
<td>41.5</td>
</tr>
<tr>
<td>ours</td>
<td>39.6</td>
<td>59.5</td>
<td>91.2</td>
<td>88.5</td>
<td>94.1</td>
<td>70.7</td>
<td>70.4</td>
<td>49.8</td>
</tr>
<tr>
<td>Improvement</td>
<td><strong>16.9</strong></td>
<td><strong>3.4</strong></td>
<td><strong>1.5</strong></td>
<td><strong>1.6</strong></td>
<td><strong>0.3</strong></td>
<td><strong>10.8</strong></td>
<td><strong>11.2</strong></td>
<td><strong>8.3</strong></td>
</tr>
<tr>
<td>Gkioxari et al. [2]</td>
<td>57.6</td>
<td>66.5</td>
<td>27.9</td>
<td>58.9</td>
<td>35.8</td>
<td>53.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wang et al. [134]</td>
<td>68.6</td>
<td>72.8</td>
<td>31.5</td>
<td>44.4</td>
<td>26.2</td>
<td>56.4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


*A: appearance model.
Table 6.5: Impact of our training data sampling strategy on video-mAP, JHMDB-21 (averaged over 3 splits).

<table>
<thead>
<tr>
<th>IoU threshold δ</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-11</td>
<td>57.73</td>
<td>57.70</td>
<td>57.60</td>
<td>56.81</td>
<td>54.27</td>
</tr>
<tr>
<td>Model-11+32</td>
<td><strong>57.79</strong></td>
<td><strong>57.76</strong></td>
<td><strong>57.68</strong></td>
<td>56.79</td>
<td><strong>55.31</strong></td>
</tr>
</tbody>
</table>

Table 6.6: Spatio-temporal action detection performance (video-mAP) comparison with the state-of-the-art on J-HMDB-21.

<table>
<thead>
<tr>
<th>IoU threshold δ</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari and Malik [2]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>53.30</td>
</tr>
<tr>
<td>Wang et al. [134]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>56.40</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>–</td>
<td>63.1</td>
<td>–</td>
<td>–</td>
<td>60.70</td>
</tr>
<tr>
<td>Saha et al. [6] (Spatial Model)</td>
<td>52.99</td>
<td>52.94</td>
<td>52.57</td>
<td>52.22</td>
<td>51.34</td>
</tr>
<tr>
<td>Peng and Schmid [11]</td>
<td>–</td>
<td><strong>74.3</strong></td>
<td>–</td>
<td>–</td>
<td><strong>73.1</strong></td>
</tr>
<tr>
<td>Ours</td>
<td>57.79</td>
<td>57.76</td>
<td>57.68</td>
<td>56.79</td>
<td>55.31</td>
</tr>
</tbody>
</table>

Table 6.7: Spatio-temporal action detection performance (video-mAP) comparison with the state-of-the-art on UCF-101.

<table>
<thead>
<tr>
<th>IoU threshold δ</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.5</th>
<th>0.75</th>
<th>0.5:0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yu et al. [30]</td>
<td>42.8</td>
<td>26.50</td>
<td>14.6</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>51.7</td>
<td>46.8</td>
<td>37.8</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Peng and Schmid [11]</td>
<td><strong>77.31</strong></td>
<td><strong>72.86</strong></td>
<td><strong>65.70</strong></td>
<td>30.87</td>
<td>01.01</td>
<td>07.11</td>
</tr>
<tr>
<td>Saha et al. [6] (*A)</td>
<td>65.45</td>
<td>56.55</td>
<td>48.52</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Saha et al. [6] (full)</td>
<td>76.12</td>
<td>66.36</td>
<td>54.93</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Ours – ML</td>
<td>68.85</td>
<td>60.06</td>
<td>49.78</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Ours – ML – (⋆)</td>
<td>70.71</td>
<td>61.36</td>
<td>50.44</td>
<td>32.01</td>
<td>0.4</td>
<td>9.68</td>
</tr>
<tr>
<td>Ours – 2PDP – (⋆)</td>
<td>71.3</td>
<td>63.06</td>
<td>51.57</td>
<td>33.06</td>
<td>0.52</td>
<td><strong>10.72</strong></td>
</tr>
</tbody>
</table>

(*) cross validated alphas as in [6]; 2PDP - tube generation algorithm [6]; ML - our micro-tube linking algorithm.

### 6.7.4 Detection performance comparison with the state-of-the-art

Table 6.6 reports action detection results, averaged over the three splits of J-HMDB-21, and compares them with those to our closest competitors. Note that, although our model only trained using the appearance features (RGB images), it outperforms [2] which was trained using both appearance and optical flow features. Also, our model outperforms [6]’s spatial detection network.

Table 6.7 compares the action detection performance of our model on the UCF-101 dataset to that of current state of the art approaches. We can ob-
serve that our model outperforms \cite{3,6,30} by a large margin. In particular, our appearance-based model outperforms \cite{3} which exploits both appearance and flow features. Also notice, our method works better than that of \cite{11} at higher IoU threshold, which is more useful in real-world applications.

### 6.8 Supporting experiments and discussion

#### 6.8.1 Impact of different fusion methods

In Table 6.8 we report video-mAPs obtained using mean and sum fusion methods for J-HMDB-21 dataset. We train our model on the combined trainset (set-11+32) (\S Section 6.6.1 and 6.7). We train two models, one using mean and another using sum fusion and denote these two models in Table 6.8 as Model-11+32 (mean-ML) and Model-11+32 (sum-ML) respectively. Action-tubes are constructed using our micro-tube linking (ML) algorithm. We can observe that at higher IoU threshold \( \delta = 0.5 \), the sum fusion performs better and improve the mAP by almost 1%. As a future work, we would like to explore different spatial and temporal feature map fusion functions \cite{40}.

Table 6.8: Impact of element-wise mean and sum fusion methods on video-mAP for J-HMDB-21 dataset (averaged over 3 splits).

<table>
<thead>
<tr>
<th>IoU threshold ( \delta )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-11+32 (mean-ML)</td>
<td>57.16</td>
<td>57.14</td>
<td>57.00</td>
<td>56.13</td>
<td>54.51</td>
</tr>
<tr>
<td>Model-11+32 (sum-ML)</td>
<td>57.79</td>
<td>57.76</td>
<td>57.68</td>
<td>56.79</td>
<td>55.31</td>
</tr>
</tbody>
</table>

#### 6.8.2 Impact of the number of predicted 3D proposals

To investigate the effect of the number of predicted 3D proposals on detection performance, we generate video-mAPs using two different sets of detections on J-HMDB-21 dataset. One detection set is generated by selecting top 1000 3D proposals and another set is by selecting top 300 3D proposals at test time using NMS. Once the two sets of detections are extracted, predicted micro-tubes are then linked up in time to generate final action tubes. Subsequently, video-mAPs are computed for each set of action tubes. The corresponding video-mAPs for each detection set at different IoU thresholds are reported in Table 6.9. We denote these two detection sets in Table 6.9 as Detection-1000 and Detection-
It is quite apparent that reduced number of RPN proposals does not effect the detection performance.

Table 6.9: Impact of the number of predicted 3D proposals on video-mAP for J-HMDB-21 dataset (averaged over 3 splits).

<table>
<thead>
<tr>
<th>IoU threshold δ</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection-1000</td>
<td>57.79</td>
<td>57.76</td>
<td>57.68</td>
<td>56.79</td>
<td>55.31</td>
</tr>
<tr>
<td>Detection-300</td>
<td>57.91</td>
<td>57.89</td>
<td>57.84</td>
<td>56.87</td>
<td>55.26</td>
</tr>
</tbody>
</table>

6.8.3 Loss function hyper-parameters

We have four hyper-parameters $\lambda_{cls}^e$, $\lambda_{loc}^e$, $\lambda_{cls}^m$ and $\lambda_{loc}^m$ in our multi-task loss function (§ Equation 6.2) which weigh the relative importance of the four loss terms. To investigate the effect of these hyper-parameters on video-mAP, we train our model with different combinations of these four hyper-parameters on J-HMDB-21 split-1. The trainset is generated as per scheme-11 (§ Section 6.6.1). The video-mAPs of these trained models are presented in Table 6.10. We can observe that when the weights for the mid classification ($\lambda_{cls}^m$) and regression ($\lambda_{loc}^m$) layers’ loss terms are too low (e.g. 0.1 & 0.05), the model has the worst detection performance. When all weights are set to 1, then the model exhibits good detection performance. However, we get the best video-mAPs with $\lambda_{cls}^e = 1.0$, $\lambda_{loc}^e = 1.0$, $\lambda_{cls}^m = 0.5$ and $\lambda_{loc}^m = 0.5$. In all our experiments we set all 4 weights to 1. As a future work, we will explore the setting [1.0, 1.0, 0.5, 0.5].

Table 6.10: Impact of different combinations of hyper-parameters on video-mAP for J-HMDB-21 split-1 train set.

<table>
<thead>
<tr>
<th>Hyper-parameters $\lambda_{cls}^e$ $\lambda_{loc}^e$ $\lambda_{cls}^m$ $\lambda_{loc}^m$</th>
<th>IoU threshold δ</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 1.0 0.1 0.05</td>
<td></td>
<td>55.03</td>
<td>55.03</td>
<td>54.63</td>
<td>53.17</td>
<td>50.33</td>
</tr>
<tr>
<td>1.0 1.0 0.1 0.1</td>
<td></td>
<td>55.62</td>
<td>55.62</td>
<td>55.47</td>
<td>54.47</td>
<td>50.51</td>
</tr>
<tr>
<td>1.0 1.0 0.5 0.25</td>
<td></td>
<td>56.30</td>
<td>56.30</td>
<td>55.91</td>
<td>54.76</td>
<td>52.30</td>
</tr>
<tr>
<td>1.0 1.0 0.5 0.5</td>
<td></td>
<td><strong>57.30</strong></td>
<td><strong>57.13</strong></td>
<td><strong>56.79</strong></td>
<td><strong>55.82</strong></td>
<td><strong>53.81</strong></td>
</tr>
<tr>
<td>1.0 1.0 1.0 1.0</td>
<td></td>
<td>56.86</td>
<td>56.85</td>
<td>56.57</td>
<td>55.89</td>
<td>52.78</td>
</tr>
</tbody>
</table>

6.8.4 Ablation study

An ablation study of the proposed model is presented in Section 6.8.6. Besides, as a part of the ablation study, per class frame- and video-APs of J-HMDB-
21 dataset are reported in Table 6.3 and per class video-APs of UCF-101 are presented in Table 6.4.

Table 6.11: An ablation study on J-HMDB-21 (split-01). Video-mAP is computed at IoU threshold $\delta = 0.5$.

<table>
<thead>
<tr>
<th>Model</th>
<th>video-mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-01</td>
<td>48.9</td>
</tr>
<tr>
<td>Model-02</td>
<td>52.7</td>
</tr>
<tr>
<td>Model-03</td>
<td>57.1</td>
</tr>
</tbody>
</table>

Model-01: Training pairs with identical frames  
Model-02: Training pairs with consecutive frames (model-11)  
Model-03: Training pairs with mixture of consecutive and successive frames (model-11+32)

6.8.5 Computing time required for training/testing

Computing time required for training. Saha et al. reported [155] that the state-of-the-art [2,3] action detection methods require at least 6+ days to train all the components (including fine-tuning CNNs, CNN feature extraction, one vs rest SVMs) of their detection pipeline for UCF-101 trainset (split-01). In our case, we need to train the model once which requires 96 hours for UCF-101 and 36 hours for J-HMDB-21 to train. The training and test time calculations are done considering a single NVIDIA Titan X GPU. The computing time requirement for different detection methods are presented in Table 6.12. Our model requires 2 days less training time as compared to [2,3] on UCF-101 trainset.

Computing time required for testing. We compare video-level computing time required (during test time) of our method with [2,3,6] on J-HMDB-21 dataset. Note that our method takes the least computing time of 8.5 Sec./video as compared to [2,3,6] (* Table 6.12).

Table 6.12: Computing time comparison for training and testing.

<table>
<thead>
<tr>
<th>Methods</th>
<th>days (*)</th>
<th>Sec/video (**)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al.</td>
<td>6+</td>
<td>113.52</td>
</tr>
<tr>
<td>Weinzaepfel et al.</td>
<td>6+</td>
<td>52.23</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>3+</td>
<td>10.89</td>
</tr>
<tr>
<td>ours</td>
<td>4</td>
<td>8.5</td>
</tr>
</tbody>
</table>

(*) Training time on UCF-101 dataset.  
(**) Average detection time on J-HMDB-21.
6.8.6 Discussion

This chapter is about action detection, where evaluation is by class-wise average precision (AP) rather than classification accuracy, a confusion matrix cannot be used. Our model is not limited to learn from pairs of consecutive frames, but can learn from pairs at any arbitrary interval $\Delta$ (see Figure 6.2 (a)).

To confute this point we conducted an ablation study of our model which is discussed below. For consecutive frames, we trained our model on J-HMDB-21 (split-01) dataset by passing training pairs composed of identical frames, e.g. passing the video frame pair (65, 65) instead of (65, 66). As you can see in Table 6.11, video-mAP drops significantly by 8.13% (at IoU threshold $\delta = 0.5$) which implies that the two streams do not output identical representations.

To double-check, we also extracted the two VGG-16 conv feature maps (see Figure 6.1 (b)) for each test frame pair $((f_t, f_{t+1}))$ of J-HMDB-21 and UCF-101 datasets. For each pair of conv feature maps, we first flattened them into feature vectors, and then computed the normalised $L_2$ distance between them. For identical frames we found that the $L_2$ distance is 0 for both J-HMDB-21 and UCF-101 datasets. Whereas, for consecutive frames it is quite high, in case of J-HMDB-21 the mean $L_2$ distance is 0.67; for UCF-101 the mean $L_2$ distance is 0.77 which again implies that the two streams generate significantly different feature encoding even for pairs consist of consecutive video frames.

6.8.7 Qualitative results

Spatiotemporal action detection results on UCF-101. We show the spatiotemporal action detection qualitative results in Figures 6.5 and 6.6. To demonstrate the robustness of the proposed detector against temporal action detection, we select those action categories which have highly temporally untrimmed videos. We select action classes VolleyballSpiking, BasketballDunk and CricketBowling. For VolleyballSpiking class, the average temporal extent of the action in each video is 40%, that means, the remaining 60% of the video does not contain any action. Similarly, for BasketballDunk and CricketBowling classes, we have average durations 41% and 46% respectively.

Video clip (a) (§ Figures 6.5) has duration 107 frames and the action VolleyballSpiking takes place only between frames 58 to 107. Note that our method able to successfully detect the temporal extent of the action (alongside spatial locations) which closely matches the ground-truth. We can observe similar quality of detection results for video clip (b) and (c) (§ Figures 6.5) which have dura-
tions 41 and 94 frames and the temporal extent of action instances are between frames 17 to 41 and frames 75 to 94 respectively for BasketballDunk and CricketBowling. Video clips (a) and (b) in Figures 6.6 show some more spatiotemporal detection results for action classes BasketballDunk and CricketBowling.

Figures 6.7 shows sample detection results on UCF-101. Note that in (1), the 2nd “biker” is detected in spite of partial occlusion. Figures 6.7 (1), (2), (3) and (5) are examples of multiple action instance detection with complex real world scenarios like 3 fencers (§ (2)) and bikers (§ (3)). Further, note that the detector is robust against scale changes as the 3rd fencer (§ (2)) and the biker (§ (3)) are detected accurately in spite of their relatively smaller shapes.

Spatiotemporal action detection results on J-HMDB-21. Figure 6.8 presents the detection results of our model on J-HMDB-21 dataset. In Figure 6.8 (1), (2) and (3), the actions “run” and “sit” are detected accurately in spite of large variations in illumination conditions, which shows that our detector is robust against illumination changes. In Figure 6.8 (5), (6) and (7), the actions “jump” and “run” are detected successfully. Note that due to fast motion, these video frames are affected by motion blur. Further, in Figure 6.8 (9) to (12), actions “stand” and “sit” are detected with correct action labels. Even for human, it is hard to infer which instance belong to “stand” and “sit” class. This again tells that our classifier is robust against inter-class similarity.

6.9 Implementation details

We implement our method using Torch 7 [156]. To develop our codebase, we take coding reference from the publicly available repository [157]. We use the coding implementation of bilinear interpolation [158] (§ Section 6.3.4) for ROI feature pooling. Our micro-tube linking algorithm (ML) (§ Section 6.5) is implemented in MATLAB.

In all our experiments, at training time we pick top 2000 RPN generated 3D proposals using NMS (non-maximum suppression). At test time we select top 1000 3D proposals. However, a lower number of proposals, e.g. top 300 proposals does not effect the detection performance, and increase the test time detection speed significantly. In Section 6.8.2 we show that extracting less number of 3D proposals (at test time) does not effect the detection performance. Shaoqing et al. [7] observed the same with Faster-RCNN.
For UCF-101, we report test time detection results (video-mAP) using two different action-tube generation algorithms. Firstly, we link the micro-tubes predicted by the proposed model (at test time) using our micro-tube linking (ML) algorithm (§ Section 6.5). We denote this as “Ours-ML” in Table 6.7. Secondly, we construct final action-tubes from the predicted micro-tubes using the 2 pass dynamic programming (2PDP) algorithm proposed by [6]. We denote this as “Ours-2PDP” in Table 6.7. The results in Table 6.1, 6.4, 6.5 and 6.6 are generated using our new micro-tube linking algorithm (“Ours-ML”). Further, we cross-validate the class-specific $\alpha_c$ as in Section 3.4 of [6], and generate action-tubes using these cross-validated $\alpha_c$ values. We denote the respective results using an asterisk (‘*’) symbol in Table 6.7.

6.9.1 Mini-batch sampling

In a similar fashion [5], we construct our gradient descent mini-batches by first sampling $N$ pairs of successive video frames, and then sampling $R$ 3D proposals for each pair. In practice, we set $N = 1$ and $R = 256$ in all our experiments. We had one concern over this way of sampling training examples because, all the positive 3D proposals from a single training batch (i.e., a pair of video frames) belong to only one action category (that is, they are correlated), which may cause slow training convergence. However, we experience a fast training convergence and good detection results with the above sampling strategy.

6.9.2 Data preprocessing

The dimension of each video frame in both J-HMDB-21 and UCF-101 is $[320 \times 240]$. We scale up each frame to dimension $[800 \times 600]$ as in [7]. Then we swap the RGB channels to BGR and subtract the VGG image mean $\{103.939, 116.779, 123.68\}$ from each BGR pixel value.

6.9.3 Data augmentation

We augment the training sets by flipping each video frame horizontally with a probability of 0.5.

---

3Each video clip of UCF-101 and J-HMDB-21 is associated with a single class label. Therefore, a pair of video frames belongs to a single action class.
6.9.4 Training batch

Our training data loader script constructs a training batch which consists of: a) a tensor of size $[2 \times D \times H \times W]$ containing the raw RGB pixel data for a pair of video frames, where $D = 3$ refers to the 3 channel RGB data, $H = 600$ is the image height and $W = 800$ is the image width; b) a tensor of size $[2 \times T \times 6]$ which contains the ground-truth micro-tube annotation in the following format: $[\text{fno}, \text{tid}, x_c, y_c, w, h]$, where $T$ is the number of micro-tubes, $\text{fno}$ is the frame number of the video frame, $\text{tid}$ is an unique identification number assigned to each individual action tube within a video, $\{x_c, y_c\}$ is the center and $w$ and $h$ are the width and height of the ground-truth bounding box; c) a $[1 \times T]$ tensor storing the action class label for each micro-tube. The J-HMDB-21 (Model-11+32) train set has 58k training batches, and UCF-101 train set consists of 340k training batches.

6.9.5 Training iteration

Our model requires at least 2 training epochs because, in the first training epoch we freeze the weights of all the convolutional layers and only update the weights of the rest of the network. We start updating the weights of the convolutional layers (alongside other layers) in the second epoch. We stop the training after 195k and 840k iterations for J-HMDB-21 and UCF-101 respectively. The training times required for J-HMDB-21 and UCF-101 are 36 and 96 GPU hours respectively using a single GPU. The training time can be further reduced by using two or more GPUs in parallel.

6.10 Fusion methods

A fusion function $f : \vec{x}_t, \vec{x}_{t+\Delta} \rightarrow y$ fuses two convolution feature maps $\vec{x}_t, \vec{x}_{t+\Delta} \in \mathbb{R}^{H' \times W' \times D}$ to produce an output map $y \in \mathbb{R}^{H' \times W' \times D}$, where $W'$, $H'$ and $D$ are the width, height and number of channels of the respective feature maps [7]. In this work we experiment with the following two fusion methods.

**Sum fusion.** Sum fusion $y_{\text{sum}} = f_{\text{sum}}(\vec{x}_t, \vec{x}_{t+\Delta})$ computes the sum of the two feature maps at the same spatial locations, $(i, j)$ and feature channels $d$:

$$y_{i,j,d}^{\text{sum}} = \vec{x}_{i,j,d}^t + \vec{x}_{i,j,d}^{t+\Delta} \quad (6.4)$$
where $1 \leq i \leq H', 1 \leq j \leq W', 1 \leq d \leq D$ and $\vec{x}^t, \vec{x}^{t+\Delta}, y \in \mathbb{R}^{H' \times W' \times D}$.

**Mean fusion.** Mean fusion is same as sum fusion, only the difference is, instead of computing the element-wise sum, here we compute the element-wise mean:

$$y^\text{mean}_{i,j,d} = (\vec{x}^t_{i,j,d} + \vec{x}^{t+\Delta}_{i,j,d}) / 2 \quad (6.5)$$
Frame No. 1 Frame No. 38 Frame No. 40 Frame No. 62
Frame No. 70 Frame No. 80 Frame No. 94 Frame No. 107
Frame No. 1 Frame No. 20 Frame No. 30 Frame No. 44
Frame No. 75 Frame No. 77 Frame No. 90 Frame No. 94
Video clip (b)
Video clip (c)
Figure 6.5: Spatiotemporal action detection results. Video clips (a), (b) and (c) are test videos belong to UCF-101 action classes VolleyballSpiking, BasketballDunk and CricketBowling, respectively.
Figure 6.6: Spatiotemporal action detection results. Video clips (a) and (b) are test videos belong to UCF-101 action classes BasketballDunk and CricketBowling respectively.

Figure 6.7: More sample detection results on UCF-101 test videos.
Figure 6.8: Spatiotemporal action detection results on J-HMDB-21 test videos.
Chapter 7


[Fabio, please give a pass to this entire chapter.]

7.1 Introduction

Following from the previous chapter (Chapter 6), here we extend the AMTnet deep architecture to improve both action detection performance and speed. The main contributions of this chapter is as follows:

- We significantly improve the action detection performance of AMTnet by incorporating optical flow features (cf. Section 7.5.1). We refer this new deep network as “AMTnet-Flow” (cf. Section 7.2).

- Unlike AMTnet, we train AMTnet-Flow on both short and long distance frame pairs, and at test time, we improve the detection speed significantly by extracting action micro-tubes on long distance pairs (cf. Section 7.4). At test time to extract longer detection micro-tubes, we propose an efficient “box interpolation algorithm” (cf. Section 7.3.1) which populates the missing bounding boxes for intermediate frames by using linear interpolation of box coordinates. For temporal linking of longer micro-tubes (cf. Section 7.3.2), we modify the micro-tube linking algorithm (Section 6.5) which further speeds up the tube generation step (cf. Section 7.5.3).

- Unlike the test time fusion approach [2, 3, 6, 11], our new train time CNN
feature fusion scheme allow the network to learn the pixel-wise correspondences between appearance and motion features [40].

**Figure 7.1: Overview of the proposed AMTnet-Flow action detection network.**

**Outline.** Coming soon ...]

**Related publication.** In this chapter, we refer the works presented in previous chapters (Chapter 4 & 6) with their related BMVC 2016 and ICCV 2017 publications [6][10] respectively.

### 7.2 Network architecture

The network architecture used in this chapter is similar to AMTnet architecture (Chapter 6). We extend AMTnet by adding an additional motion stream alongside the existing two RGB flow streams. We use VGG-16 netowrk architecture [31] for our two RGB and one flow stream (Figure 7.1 (b.1) & (b.2)). We keep all the layers of VGG-16 from the first layer to the last convolutional layer, and discard the remaining layers. In this chapter we call this modified VGG as VGG-16*.

#### 7.2.1 Appearance streams

We use two parallel VGG-16* for our appearance streams (Figure 7.1 (b.1)). At each forward pass (during training and testing), these two CNNs receive
two successive RGB video frames \( f_t \) and \( f_{t+\Delta} \) (Figure 7.1 (a.1)) and process them through 13 convolutional layers (intermixed with 5 max pooling layers) in parallel. The output of each CNN is a convolutional feature map produced by the last conv layer which has dimension \([D \times H' \times W']\) where \( D = 512 \), \( H' = \frac{H}{16} \), and \( W' = \frac{W}{16} \). \( H \) is the input frame height, \( W \) is the frame width.

### 7.2.2 Motion stream

We use a single VGG-16\( ^* \) for the motion stream (Figure 7.1 (b.2)). The motion stream takes 5 stacked optical flow maps (Figure 7.1 (a.2)) \(^{[11]} \) as input. It processes the stacked flow maps in parallel to the appearance stream, and outputs a feature map of dimension \([D \times H' \times W']\). The 3 feature-maps are then fused using a sum fusion \(^{[40]}\) and the resultant feature-map (Figure 7.1 (c)) is passed as input to a 3D-RPN.

Please refer Section 6.3.2 and 6.3.3 for 3D-RPN and 3D-RoI sampler respectively (Figure 7.1 (d)), Section 6.3.4 for RoI feature pooling (using bilinear interpolation) and fusion (Figure 7.1 (e)).

### 7.2.3 Fully connected, classification and regression layers

The features (Figure 7.1 (f)) of each positive and negative 3D-ROIs are processed through two fully connected (fc6 and fc7), a classification and a regression layers (Figure 7.1 (g)) to train for action classification and micro-tube regression. The outputs of the network are \( C \times M \) softmax classification scores and \( 8 \times M \) bounding box coordinates for \( M \) predicted micro-tubes (Figure 7.1 (h)) and \( C \) action categories. In all our experiments, at train time we use \( M = 2000 \) and test time \( M = 300 \).

### 7.3 Bounding box interpolation and action tube generation

At test time, for a input pair of successive video frames \( (f_t \) and \( f_{t+\Delta} \)), the network outputs action classification scores and coordinate values for 300 micro-tubes (Section 7.2.3). For example, in Figure 7.2 (a), we can visualise that for an input pair \( (F1, F4) \) our network predicts an action micro-tube (depicted by the 2 blue bounding boxes) with 8 coordinate values (blue circles). In practice, the
network predicts such 300 micro-tubes for an input pair. Note that, yet we do not have detection bounding boxes for intermediate frames, in this example those are frames $F_2$ and $F_3$. We generate the detection boxes for the intermediate frames using a simple but elegant box interpolation algorithm which is explained in the next section.

Figure 7.3: Generation of detection bounding boxes for intermediate frames using linear interpolation.

### 7.3.1 Bounding box interpolation

As depicted in Figure 7.3, the 8 predicted bounding box coordinates $[(x_1, y_1, x_2, y_2)^{t_1}]$ and $[(x_1, y_1, x_2, y_2)^{t_4}]$ (shown in blue) which are known, and we would like to infer the unknowns $[(x_1, y_1, x_2, y_2)^{t_2}]$ and $[(x_1, y_1, x_2, y_2)^{t_3}]$ (shown in pink). Using simple linear interpolation we can compute the unknowns using the following equation:

$$
x_1^{t_2} = x_1^{t_1} \left(1 - \frac{t_2 - t_1}{t_4 - t_1}\right) + x_1^{t_4} \left(1 - \frac{t_2 - t_1}{t_4 - t_1}\right) \quad (7.1)
$$

$$
y_1^{t_2} = y_1^{t_1} \left(1 - \frac{t_2 - t_1}{t_4 - t_1}\right) + y_1^{t_4} \left(1 - \frac{t_2 - t_1}{t_4 - t_1}\right) \quad (7.2)
$$

Figure 7.2: (a) Action micro-tube generation by linear interpolation of coordinates of predicted bounding boxes. The blue circles denote the 8 $(x,y)$ coordinate values of bounding boxes predicted by our network for a pair of successive test video frames. The first frame in the pair belongs to time $t = 1$ and the seconds frame belongs to time $t = 4$. We generate the coordinates of detection bounding boxes (pink circles) for intermediate frames (i.e. frame 2 and 3) by linear interpolation of the predicted coordinates. Thus, at test time, our model processes a pair of successive frames by skipping two intermediate frames (in this example frame 2 and 3) which substantially reduces the computing cost and time and speeds up the detection process. (b) Action tube generation by linking micro-tubes generated by (a). Note that, for a video sequence with $F$ frames, our model needs to process only $F/2$ frames and the micro-tube linking algorithm requires to connect only $(F/2) - 2$ frames.
Similarly we can compute $(x_2, y_2)^{t_2}$ and $(x_2, y_2)^{t_3}$.

### 7.3.2 Action tube generation

Once the bounding-boxes for intermediate frames are generated using the linear interpolation algorithm (§7.3.1) for each micro-tubes, we can then temporally link them (Figure 7.2 (b)) using action tube generation algorithm (cf. Section 6.5) presented in previous chapter.

The micro-tube linking algorithm presented in Section 6.5 requires lesser iterations than the one in Section 6.3.4 i.e., it requires $(F/2 - 1)$ as compared to $(F - 1)$ iterations of Section 6.3.4 (where $F$ is the number of frames in a video). Unlike in Section 6.5 where micro-tubes are extracted for consecutive frame pairs (i.e. $\Delta = 1$), in this chapter, we use long distance frame pairs (i.e. $\Delta = 3$) at test time which allow our micro-tube linking algorithm to iterate only $(F/4 - 1)$ times to link all the micro-tubes associated with an action instance within a video. Thus, this new tube generation approach is relatively faster and requires less compute time and resources than the one in Section 6.5. For instance, as depicted in Figure 7.2 (b), tube generation algorithm presented here requires only 3 iterations to link the micro-tubes of a video having 16 frames. Whereas, the algorithm in Section 6.3.4 needs 15 iterations and the one in Section 6.5 requires 7 iterations to link them in time.

We extend the action tube generation (i.e. micro-tube linking) module presented in Section 6.5. In the 1st pass of DP (dynamic programming), class-specific temporally untrimmed action tubes $u_c = \{m_i, \quad i \in I = \{4, 8, ..., T-4\}\}$, spanning the entire video duration are obtained by maximising via dynamic programming:

$$E(u_c) = \sum_{i \in I} s_c(m_i) + \lambda \sum_{i \in I} \phi(b_{m_i}^2, b_{m_i+1}^1),$$  \hspace{1cm} (7.5)

where $m$ denotes the micro-tube having two bounding boxes $b^1$ and $b^2$, $s_c(m_i)$ represents the softmax score (§ Section 7.2.3) of the predicted micro-tube $m$ at time step $i$, $\phi(b_{m_i}^2, b_{m_i+1}^1)$ is the IoU (intersection over union) between the 2nd detection box $b_{m_i}^2$ of micro-tube $m_i$ and the first detection box $b_{m_i+1}^1$ of micro-tube $m_{i+1}$. $\lambda$ is a scalar parameter weighting the relative importance of the pairwise term. The above energy maximisation (§ Eq. 7.5) gives us action tubes

which are temporally untrimmed. For temporal localisation of actions, we trim the action tubes using a similar approach as in Section 6.5 using a $2^{dn}$ pass of DP (or a temporal label smoothing approach).

### 7.4 Train and test data sampling schemes

To train and test our proposed model, we need to pass pairs of successive RGB video frames ($f_t, f_{t+\Delta}$) and a set of 5 stacked optical flow maps as inputs to the network. At training time, we generate long- and short-distance training pairs using different $\Delta$ values.

Figure 7.4 shows four different data sampling schemes, in which $F_i$ denotes a RGB video frame where $i$ is the frame index and $P_j$ denotes a training pair where $j$ is the pair index. $\text{Scheme-11}$ and $\text{scheme-21}$ ($\S$ Figure 7.4(a) & (b)) generate short-distance pairs using $\Delta$ value 1. Whereas, $\text{Scheme-32}$ and $\text{scheme-43}$ ($\S$ Figure 7.4(c) & (d)) generate relatively long-distance pairs using $\Delta$ values 2 and 3 respectively.

#### 7.4.1 Training data of RGB video frames

To training our model on $J$-HMDB-21 dataset, we generate train sets using data sampling schemes: $\text{scheme-11}$, -32 and -43 which give rise to a total 68796, 69350 and 69054 training pairs for split 1, 2 and 3 respectively. For $UCF-101-24$ dataset (split-1), we create a training set by generating pairs using $\text{scheme-21}$, -43 which jointly give 509,940 training pairs.
7.4.2 Training data of optical-flow maps

We capture the local motion patterns (of human actions) between two consecutive video frames (cf. Section 1.5.7) by densely sampling optical-flow fields between two consecutive pairs of video frames using the widely used optical-flow algorithm [47]. Then, we represent the optical-flow fields between two consecutive frames as a 3 channel RGB image [2] which we call “flow-map”, in which the first 2 channels represent the flow in $x$ and $y$ directions respectively, and the 3rd channel represents the magnitudes of the flow vectors.

For training pairs generated using scheme-11 and -21 ($\Delta = 1$) (§ Section 7.4.1), we select the corresponding five flow-maps following a sequence:

\[\{f_{t-1}, f_t, f_{t+\Delta}, f_{t+\Delta+1}, f_{t+\Delta+2}\},\]

designed to ensure a smooth transition in motion, as dictated by the sampling scheme.

For scheme-32 ($\Delta = 2$), we follow a sequence:

\[\{f_{t-1}, f_t, f_{t+1}, f_{t+\Delta}, f_{t+\Delta+1}\},\]

and for scheme-43 ($\Delta = 3$):

\[\{f_{t-1}, f_t, f_{t+1}, f_{t+2}, f_{t+\Delta}\}.\]

7.4.3 Test time data sampling

At test time, we sample long-distance RGB frame pairs using scheme-43 (§ Section 7.4.1) and follow the corresponding sequence (§ Section 7.4.2) to select five optical-flow maps.

7.5 Model Evaluation

Evaluation metrics. As evaluation metrics we use the standard frame-mAP [2, 11] and video-mAP [2, 3, 6, 11] (mean average precision). For video-mAP evaluation, we select the IoU threshold ($\delta$) range [0.1 : 0.1 : 0.5]. For UCF-101-24, we compute the video-mAPs at higher IoU threshold range between [0.5 : 0.05 : 0.95], and then we compute the mean video-mAP which we denote as (0.5 : 0.95). For frame-mAP we set IoU threshold $\delta = 0.5$ for both J-HMDB-21 and UCF-101-24.

Experimental settings. We follow the training data sampling strategy as in [10], in addition, for training on long distance pairs we introduce a new scheme scheme-43. Throughout this section we use the following naming conventions to denote models trained on training sets generated using different sampling strategies: a) a model trained on training data generated using scheme-11 and -

### Table 7.1: Impact of adding motion stream on action detection performance, J-HMDB-21 (*) dataset.

| δ   | 0.1  | 0.2  | 0.3  | 0.4  | 0.5  | 0.5  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AMTnet</td>
<td>57.79</td>
<td>57.76</td>
<td>57.68</td>
<td>56.79</td>
<td>55.31</td>
<td>45.0</td>
</tr>
<tr>
<td>AMTnet-Flow</td>
<td><strong>64.64</strong></td>
<td><strong>64.64</strong></td>
<td><strong>64.57</strong></td>
<td><strong>64.01</strong></td>
<td><strong>62.68</strong></td>
<td><strong>52.08</strong></td>
</tr>
</tbody>
</table>

(*)Trained model-11+32, action tubes generated using micro-tube linking algorithm as in [10].

### Table 7.2: Impact of adding motion stream on action detection performance, UCF-101-24 (*) dataset.

| δ   | 0.1  | 0.2  | 0.3  | 0.4  | 0.5  | 0.5  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AMTnet</td>
<td>71.3</td>
<td>63.06</td>
<td>51.57</td>
<td>43.10</td>
<td>33.06</td>
<td>54.91</td>
</tr>
<tr>
<td>AMTnet-Flow</td>
<td><strong>79.65</strong></td>
<td><strong>71.51</strong></td>
<td><strong>59.79</strong></td>
<td><strong>47.28</strong></td>
<td><strong>35.77</strong></td>
<td><strong>62.70</strong></td>
</tr>
</tbody>
</table>

(*)Trained model-21, action tubes generated using 2PDP with cross validated alphas as in [6].

32 [10] is referred as *model-11+32*; b) *model-11+32+43* refers the model trained using *scheme-11, -32 and -43*; similarly, c) *model-21* and d) *model-21+43*. For more details on the train and test data sampling schemes please refer Section 7.4.

#### 7.5.1 Impact of adding motion stream on action detection performance

We observe a significant improvement in action detection performance by adding an extra flow stream to the AMTnet architecture. For this experiment, we first reproduce the results of AMTnet [10] on both J-HMDB-21 and UCF-101-24 datasets following their train and test time settings. Then, we introduce our motion-stream alongside the two spatial streams of AMTnet, and train the model on both the datasets following AMTnet’s train and test time settings. In Table 7.1 and 7.2, we refer these models as AMTnet and AMTnet-Flow where the video- and frame-mAPs at different IoU thresholds (δ) for these two models are shown. Note that for J-HMDB, the video- and frame-mAP are improved by **7.37%** and **7.08%** at δ = 0.5 (Table 7.1). For UCF-101-24, video-mAP (at δ = 0.2) and frame-mAP (at δ = 0.5) are improved by **8.45%** and **7.79%** (Table 7.2).
Table 7.3: Impact of bounding box interpolation on action detection performance (video-mAP).

<table>
<thead>
<tr>
<th>δ</th>
<th>J-HMDB</th>
<th>UCF-101</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>64.73</td>
<td>71.57</td>
</tr>
<tr>
<td>0.4</td>
<td>64.32</td>
<td>35.77</td>
</tr>
<tr>
<td>0.5</td>
<td>62.67</td>
<td>10.9</td>
</tr>
<tr>
<td>w/o interpolation</td>
<td>64.63</td>
<td>69.86</td>
</tr>
<tr>
<td>with interpolation</td>
<td>64.27</td>
<td>37.15</td>
</tr>
</tbody>
</table>

Table 7.4: Impact of bounding box interpolation on action detection speed (seconds / video).

<table>
<thead>
<tr>
<th></th>
<th>J-HMDB</th>
<th>UCF-101</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o interpolation</td>
<td>11.9</td>
<td>61.25</td>
</tr>
<tr>
<td>with interpolation</td>
<td>5.95</td>
<td>30.62</td>
</tr>
</tbody>
</table>

7.5.2 Impact of bounding box interpolation on detection performance and speed

For this experiment, we train our model on J-HMDB-21 and UCF-101-24 datasets on both short and long distance pairs. In Section 7.4 we discuss in detail the different training data sampling schemes used to generate the training data for each model in this experiment. At test time, we generate two sets of detection results to demonstrate the efficacy of our bounding box interpolation algorithm (§7.3). First set of micro-tubes are extracted by passing test pairs \( f_t \) and \( f_{t+\Delta} \) where \( \Delta = 1 \) (short distance pairs). Second set of micro-tubes are extracted by passing test pairs where \( \Delta = 3 \) (long distance pairs). For micro-tubes extracted using long-distance pairs, we fill in the intermediate frames’ bounding boxes using our box interpolation algorithm (§7.3). We denote these two sets of results in Table 7.3 and Table 7.4 as “w/o interpolation” and “with interpolation”. Our bounding box interpolation algorithm improves the test time detection speed significantly without affecting the detection performance (Table 7.4). For J-HMDB-21, it improves the detection speed from 11.9 sec/vid (seconds per video) to 5.95 sec/vid, and for UCF-101, from 61.25 sec/vid to 30.62 sec/vid. Here detection speed denotes the compute time required for micro-tube extraction. Interestingly, for UCF-101, it improves the video-mAP at higher thresholds, i.e. we observe a gain in the mAP by 1.38% and 1.0% at \( \delta = 0.5 \) and 0.5 : 0.95 respectively (Table 7.3). Note that, the test video clips in UCF-101-24 have relatively longer duration (on average 175 frames) than J-HMDB-21 (on average 25 frames). The relatively shorter test video clips might be the reason that our bounding box interpolation algorithm could not exhibit performance boost on J-HMDB-21. We believe that, it will work gracefully for
Table 7.5: Impact of our action tube generation algorithm on compute time (in seconds per video).

<table>
<thead>
<tr>
<th></th>
<th>Saha et al. [6]</th>
<th>Ours</th>
<th>Saha et al. [6]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-HMDB-21</td>
<td>0.38</td>
<td><strong>0.16</strong></td>
<td>0.96</td>
<td><strong>0.47</strong></td>
</tr>
</tbody>
</table>

Table 7.6: Comparison with the state-of-art on J-HMDB-21 dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Video-mAP (δ=0.2)</th>
<th>Frame-mAP (δ=0.5)</th>
<th>Frame-mAP (δ=0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al. [2]</td>
<td>–</td>
<td>53.3</td>
<td>36.2</td>
</tr>
<tr>
<td>Wang et al. [134]</td>
<td>–</td>
<td>56.4</td>
<td>39.9</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>–</td>
<td>60.7</td>
<td>45.8</td>
</tr>
<tr>
<td>Yu et al. [30]</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Saha et al. [6]</td>
<td>72.63</td>
<td>71.50</td>
<td>–</td>
</tr>
<tr>
<td>Saha et al. [10]</td>
<td>57.76</td>
<td>55.31</td>
<td>45.0</td>
</tr>
<tr>
<td>Peng et al. [11]</td>
<td><strong>74.3</strong></td>
<td><strong>73.1</strong></td>
<td><strong>58.5</strong></td>
</tr>
<tr>
<td>Ours (w/o interpolation)</td>
<td>64.73</td>
<td>62.67</td>
<td>52.08</td>
</tr>
<tr>
<td>Ours (w/with interpolation)</td>
<td>64.63</td>
<td>61.97</td>
<td>52.73</td>
</tr>
</tbody>
</table>

even longer sequences of videos where spatial locations of actions do not vary rapidly within shorter time span (e.g. within 5 or 10 frames).

7.5.3 Impact of our action tube generation algorithm on compute time

To evaluate our action tube generation method (§7.3), we compare the detection performance and compute time requirement of our algorithm with [6]. In Table 7.3, the “w/o interpolation” refers to the results obtained using the action-tube generation algorithm [6], and “with interpolation” denotes the results generated by our tube building approach. Remarkably, with relatively lesser number of iterations ($F/4 - 1$) as compared to ($F - 1$) iterations required by [6], our action tube generation algorithm improves the video-mAP at higher thresholds on UCF-101. Moreover, our method exhibits excellent mean detection speed (§ Table 7.5) of 0.16 sec/vid and 0.47 sec/vid for J-HMDB and UCF-101 respectively. In particular, it improves the speed by 0.22 (for J-HMDB) and 0.49 (for UCF-101) sec/vid over [6]’s approach.

7.5.4 Comparison with the state-of-art

We conclude our model evaluation with a comparison to the state-of-the-art results (see Table 7.7). The proposed model outperforms the state-of-the-art methods [2] [3] [10] [30] [134] in both frame- and video-mAP on two benchmark

Table 7.7: Comparison with the state-of-art on UCF-101-24 dataset.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>Video-mAP</th>
<th>Frame-mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Gkioxari et al. [2]</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Wang et al. [134]</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>51.7</td>
<td>46.8</td>
</tr>
<tr>
<td>Yu et al. [30]</td>
<td>42.8</td>
<td>26.5</td>
</tr>
<tr>
<td>Saha et al. [6]</td>
<td>76.57</td>
<td>66.75</td>
</tr>
<tr>
<td>Saha et al. [10]</td>
<td>71.3</td>
<td>63.06</td>
</tr>
<tr>
<td>Peng et al. [11]</td>
<td>77.31</td>
<td><strong>72.86</strong></td>
</tr>
<tr>
<td>Ours (w/o interpolation)</td>
<td><strong>79.65</strong></td>
<td>71.57</td>
</tr>
<tr>
<td>Ours (with interpolation)</td>
<td>78.24</td>
<td>69.86</td>
</tr>
</tbody>
</table>

datasets. In addition, our approach outperforms the two close competitors [6, 11] in spatiotemporal detection (video-mAP) on UCF-101. More specifically, it outperforms [6] (in video-mAP for UCF-101) by 4.82%, 1.29% and 1.18% at IoU threshold (\( \delta \)) values 0.2, 0.5 and 0.5 : 0.95 respectively. Although, on lower \( \delta \) values (0.2 and 0.3) our approach shows comparable results to [11], on higher \( \delta \)s (0.5 and 0.5 : 0.95), it achieves superior performance with a gain in the video-mAP by 6.28% and 4.79% respectively. A lower detection performance of our approach on J-HMDB-21 as compared to [6,11] is due to the fact that J-HMDB is relatively smaller dataset and our network has large number of model parameters than [6,11] which might cause overfitting. In Section 7.5.5, we present the class-specific frame- and video-mAP comparison with the state-of-art.

7.5.5 Comparison of class-specific frame- and video-level APs

In this section, we compare the class-specific frame- and video-level AP (average precision) with the state-of-the-art [2, 3, 6, 10, 11, 134]. We report frame-and video-AP on J-HMDB-21 and video-AP on UCF-101-24 datasets. In Table 7.9 and 7.10, we denote the highest APs with red and second highest APs with blue colour.

J-HMDB-21. For J-HMDB-21, both frame and video-APs are computed at IoU threshold \( \delta = 0.5 \) and results are averaged over the 3 splits of J-HMDB-21. Table 7.9 presents the per class frame-AP comparison with the state-of-the-art [2, 3, 10, 11, 134]. Our method achieves the highest frame-AP for five action categories with an improvement of 8.4%, 7.6%, 6.1%, 3.6% and 0.1% for action class climbStairs, run, catch, kickBall and shootGun respectively. For

Table 7.8: J-HMDB-21: per class frame-AP comparison with the state-of-the-art.

<table>
<thead>
<tr>
<th></th>
<th>brushHair</th>
<th>catch</th>
<th>clap</th>
<th>climbStairs</th>
<th>golf</th>
<th>jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al. [2]</td>
<td>65.2</td>
<td>18.3</td>
<td>38.1</td>
<td>39.0</td>
<td>79.4</td>
<td>7.3</td>
</tr>
<tr>
<td>Wang et al. [131]</td>
<td>60.1</td>
<td>34.2</td>
<td>56.4</td>
<td>38.9</td>
<td>83.1</td>
<td>10.8</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>73.3</td>
<td>34.0</td>
<td>40.8</td>
<td>56.8</td>
<td>93.9</td>
<td>5.9</td>
</tr>
<tr>
<td>Saha et al. [10]</td>
<td>43.7</td>
<td>43.6</td>
<td>33.0</td>
<td>61.5</td>
<td>91.8</td>
<td>5.6</td>
</tr>
<tr>
<td>Peng et al. [11]</td>
<td>75.8</td>
<td>38.4</td>
<td>62.2</td>
<td>62.4</td>
<td>99.6</td>
<td>12.7</td>
</tr>
<tr>
<td>Ours</td>
<td>50.9</td>
<td>49.7</td>
<td>34.5</td>
<td>70.8</td>
<td>93.7</td>
<td>12.2</td>
</tr>
</tbody>
</table>

Frame-AP(%) at $\delta = 0.5$.

<table>
<thead>
<tr>
<th></th>
<th>run</th>
<th>shootBall</th>
<th>shootBow</th>
<th>shootGun</th>
<th>sit</th>
<th>stand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al. [2]</td>
<td>11.6</td>
<td>5.6</td>
<td>66.8</td>
<td>27.0</td>
<td>32.1</td>
<td>34.2</td>
</tr>
<tr>
<td>Wang et al. [131]</td>
<td>19.8</td>
<td>11.6</td>
<td>78.0</td>
<td>50.6</td>
<td>10.9</td>
<td>43.0</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>21.1</td>
<td>23.9</td>
<td>85.6</td>
<td>37.8</td>
<td>34.9</td>
<td>49.2</td>
</tr>
<tr>
<td>Saha et al. [10]</td>
<td>32.3</td>
<td>33.3</td>
<td>81.4</td>
<td>55.1</td>
<td>12.4</td>
<td>14.7</td>
</tr>
<tr>
<td>Peng et al. [11]</td>
<td>38.1</td>
<td>52.8</td>
<td>90.8</td>
<td>62.7</td>
<td>33.6</td>
<td>48.9</td>
</tr>
<tr>
<td>Ours</td>
<td>45.7</td>
<td>42.6</td>
<td>88.3</td>
<td>62.8</td>
<td>23.3</td>
<td>45.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>kickBall</th>
<th>pick</th>
<th>pour</th>
<th>pullup</th>
<th>push</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al. [2]</td>
<td>9.4</td>
<td>25.2</td>
<td>80.2</td>
<td>82.8</td>
<td>33.6</td>
</tr>
<tr>
<td>Wang et al. [131]</td>
<td>24.5</td>
<td>38.5</td>
<td>71.5</td>
<td>67.5</td>
<td>21.3</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>13.8</td>
<td>38.5</td>
<td>88.1</td>
<td>89.4</td>
<td>60.5</td>
</tr>
<tr>
<td>Saha et al. [10]</td>
<td>23.8</td>
<td>31.5</td>
<td>91.8</td>
<td>84.1</td>
<td>73.1</td>
</tr>
<tr>
<td>Peng et al. [11]</td>
<td>35.1</td>
<td>57.8</td>
<td>96.8</td>
<td>97.3</td>
<td>79.6</td>
</tr>
<tr>
<td>Ours</td>
<td>38.7</td>
<td>41.9</td>
<td>92.1</td>
<td>87.4</td>
<td>75.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>swingBaseball</th>
<th>throw</th>
<th>walk</th>
<th>wave</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al. [2]</td>
<td>33.6</td>
<td>15.5</td>
<td>34.0</td>
<td>21.9</td>
<td>36.2</td>
</tr>
<tr>
<td>Wang et al. [131]</td>
<td>48.9</td>
<td>26.5</td>
<td>25.2</td>
<td>15.8</td>
<td>39.9</td>
</tr>
<tr>
<td>Weinzaepfel et al. [3]</td>
<td>36.7</td>
<td>16.8</td>
<td>40.5</td>
<td>20.5</td>
<td>45.8</td>
</tr>
<tr>
<td>Saha et al. [10]</td>
<td>56.3</td>
<td>22.2</td>
<td>24.7</td>
<td>29.4</td>
<td>45.0</td>
</tr>
<tr>
<td>Peng et al. [11]</td>
<td>62.2</td>
<td>25.6</td>
<td>59.7</td>
<td>37.1</td>
<td>58.5</td>
</tr>
<tr>
<td>Ours</td>
<td>48.4</td>
<td>23.9</td>
<td>42.5</td>
<td>24.0</td>
<td>52.7</td>
</tr>
</tbody>
</table>

another 7 action categories, it achieves the second highest AP with an improvement of 9.3%, 3.4%, 2.7%, 2%, 1.4% and 0.3% for class shootBall, pick, shootBow, walk, push, jump and pour respectively. Table ?? presents the per class video-AP comparison with [2, 10, 134]. We could not consider methods [3, 6, 11] in our class-specific video-AP comparison, as they do not report per class video-AP for J-HMDB-21 in their respective papers. Among 21 action classes of J-HMDB-21, our method shows superior video-level performance for 7 classes with an improvement of 16.5%, 13.4%, 9%, 7.4%, 3.7%, 2.5% and 1.8% for action class shootBall, climbStairs, run, catch, push, shootBow and kickBall respectively.

UCF-101-24. UCF-101-24 is a temporally untrimmed action detection dataset for which we have both spatial (frame-level bounding boxes) and temporal (start
Table 7.9: J-HMDB-21: per class video-AP comparison with the state-of-the-art.

<table>
<thead>
<tr>
<th>Action</th>
<th>Video-AP(%)(*) at δ = 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>brushHair</td>
</tr>
<tr>
<td>Gkioxari et al.</td>
<td>79.1</td>
</tr>
<tr>
<td>Wang et al.</td>
<td>76.4</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>51.9</td>
</tr>
<tr>
<td>Ours</td>
<td>65.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>run</th>
<th>shootBall</th>
<th>shootBow</th>
<th>shootGun</th>
<th>sit</th>
<th>stand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al.</td>
<td>24.6</td>
<td>13.7</td>
<td>92.9</td>
<td>42.3</td>
<td>67.2</td>
<td>57.6</td>
</tr>
<tr>
<td>Wang et al.</td>
<td>35.7</td>
<td>27.0</td>
<td>88.8</td>
<td>76.9</td>
<td>29.8</td>
<td>68.6</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>49.0</td>
<td>37.4</td>
<td>92.7</td>
<td>75.8</td>
<td>21.6</td>
<td>27.1</td>
</tr>
<tr>
<td>Ours</td>
<td>58.0</td>
<td>53.9</td>
<td>95.4</td>
<td>73.9</td>
<td>40.7</td>
<td>49.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>kickBall</th>
<th>pick</th>
<th>pour</th>
<th>pullup</th>
<th>push</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al.</td>
<td>26.2</td>
<td>42.0</td>
<td>92.8</td>
<td>98.1</td>
<td>29.6</td>
</tr>
<tr>
<td>Wang et al.</td>
<td>57.7</td>
<td>70.5</td>
<td>78.7</td>
<td>77.2</td>
<td>31.7</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>48.7</td>
<td>33.7</td>
<td>97.6</td>
<td>92.5</td>
<td>87.6</td>
</tr>
<tr>
<td>Ours</td>
<td>59.5</td>
<td>47.9</td>
<td>97.4</td>
<td>92.6</td>
<td>91.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>swingBaseball</th>
<th>throw</th>
<th>walk</th>
<th>wave</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gkioxari et al.</td>
<td>66.5</td>
<td>27.9</td>
<td>58.9</td>
<td>35.8</td>
<td>53.3</td>
</tr>
<tr>
<td>Wang et al.</td>
<td>72.8</td>
<td>31.5</td>
<td>44.4</td>
<td>26.2</td>
<td>56.4</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>73.3</td>
<td>24.3</td>
<td>37.7</td>
<td>44.7</td>
<td>55.3</td>
</tr>
<tr>
<td>Ours</td>
<td>62.6</td>
<td>27.8</td>
<td>54.8</td>
<td>43.0</td>
<td>62.67</td>
</tr>
</tbody>
</table>

Table 7.9 shows the per class video-AP comparison with the state-of-the-art. Among 24 action classes, our method achieves the highest video-AP for 13 classes with an overall gain in the video-mAP of 4.82%. Note that, our method improves the quality of the spatiotemporal action detection with a large margin. It can be observed by looking at the video-APs of those classes which contain significantly temporally untrimmed test video clips. For example, action classes VolleyballSpiking, CricketBowling and BasketballDunk have videos (in UCF-101 testsplit-1) which contain action instances less than 50% of the entire video on average. In other words, a test video belongs to these classes do not have any ac-
### Table 7.10: UCF-101-24: per class video-AP comparison with the state-of-the-art.

<table>
<thead>
<tr>
<th>Class</th>
<th>BaBa</th>
<th>BaDu</th>
<th>Bi</th>
<th>ClDi</th>
<th>CrBo</th>
<th>Di</th>
<th>Fe</th>
<th>FlGy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saha et al.</td>
<td>36.7</td>
<td>48.3</td>
<td>60.4</td>
<td>73.2</td>
<td>19.9</td>
<td>96.6</td>
<td>88.0</td>
<td>99.7</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>29.8</td>
<td>40.9</td>
<td>63.8</td>
<td>44.9</td>
<td>15.2</td>
<td>91.2</td>
<td>89.3</td>
<td>94.2</td>
</tr>
<tr>
<td>Ours</td>
<td>26.8</td>
<td>75.0</td>
<td>60.4</td>
<td>91.4</td>
<td>31.8</td>
<td>97.4</td>
<td>90.4</td>
<td>99.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>GoSw</th>
<th>HoRi</th>
<th>IcDa</th>
<th>LoJu</th>
<th>PoVa</th>
<th>RoCl</th>
<th>SaSp</th>
<th>SkBo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saha et al.</td>
<td>66.5</td>
<td>94.1</td>
<td>62.5</td>
<td>55.7</td>
<td>72.6</td>
<td>89.6</td>
<td>57.5</td>
<td>85.0</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>70.7</td>
<td>92.3</td>
<td>70.5</td>
<td>50.7</td>
<td>71.0</td>
<td>97.3</td>
<td>31.4</td>
<td>72.0</td>
</tr>
<tr>
<td>Ours</td>
<td>80.3</td>
<td>92.8</td>
<td>74.0</td>
<td>80.4</td>
<td>87.0</td>
<td>97.1</td>
<td>37.6</td>
<td>76.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Sk</th>
<th>SkJe</th>
<th>SoJu</th>
<th>Su</th>
<th>TeSw</th>
<th>TrJu</th>
<th>VoSp</th>
<th>WaDo</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saha et al.</td>
<td>78.9</td>
<td>92.8</td>
<td>86.4</td>
<td>61.3</td>
<td>32.6</td>
<td>51.3</td>
<td>15.9</td>
<td>75.6</td>
<td>66.75</td>
</tr>
<tr>
<td>Saha et al.</td>
<td>74.0</td>
<td>95.9</td>
<td>80.2</td>
<td>67.6</td>
<td>40.4</td>
<td>42.0</td>
<td>13.0</td>
<td>75.0</td>
<td>63.06</td>
</tr>
<tr>
<td>Ours</td>
<td>76.7</td>
<td>96.2</td>
<td>93.2</td>
<td>62.8</td>
<td>32.0</td>
<td>47.4</td>
<td>31.8</td>
<td>78.0</td>
<td>71.57</td>
</tr>
</tbody>
</table>


Our method improves the video-AP for these classes by a large margin of 26.7%, 15.9% and 11.9% for classes BasketballDunk, VolleyballSpiking and CricketBowling respectively. Similarly, we have another two classes CliffDiving and Diving which have highly temporally untrimmed video clips in the testset. For these two classes, we report an improvement of 18.2% and 0.8% in video-AP.

### 7.6 Implementation details

#### 7.6.1 Hardware and software platform

For all our experiments, we use an Ubuntu 14.0 LTS based Dell workstation equipped with an Intel Xeon CPU @ 3.20GHz, 64 GB of RAM and a Nvidia GeForce GTX TITAN X GPU. To implement our proposed model, we use the scientific computing framework Torch 7.0 [156] and scripting language LuaJIT [7] . We implement our action-tube generation algorithm in MATLAB. During implementation, we refer the Torch codebase developed by [10]. For bilinear interpolation, we use the publicly available Torch repository [158].
7.6.2 Data preprocessing and train data augmentation

Both at train and test time, we preprocess the training data by first scaling each video frame to $800 \times 600$ pixel resolution [7], and then subtracting the VGG image mean $\{103.939, 116.779, 123.68\}$ from each RGB frame, and optical-flow mean $\{128, 128, 128\}$ from each flow-map [11]. We use horizontal flipping of each video frame with a probability 0.5 to augment our training sets.

7.6.3 Network weight initialisation and optimisation

We initialise the weights of the convolutional layers with VGG-16 pretrained ImageNet [147] weights. For the rest of the network layers, weights are initialised by drawing random samples from a Gaussian distribution with a standard deviation 0.01. We train our network end-to-end as in [10]. At each training iteration, we update the weights of the convolutional layers of both appearance- and motion-streams using stochastic gradient descent (SGD) with a momentum 0.9; the Adam optimisation algorithm [154] is used to update the weights of the rest of the layers. For Adam, we use parameter values $\beta_1 = 0.9$, $\beta_2 = 0.99$ and a learning rate of $1 \times 10^{-6}$. In the first training epoch, we update the weights of all the network layers excluding the three parallel CNNs. During the first epoch we freeze the weights of the convolutional layers. From second epoch onward, we update the weights of all layers including the appearance- and motion-stream CNNs. For computational efficiency, the first four layers of the three CNNs are not fine-tuned.

7.6.4 Architectural modification for Motion Stream

We modify the architecture of the $1^{st}$ convolutional layer of VGG-16, to pass five stacked optical-flow maps as inputs to the motion stream. The original VGG-16 can receive input a tensor of shape $[3 \times H \times W]$, i.e., a single video frame ($H$ and $W$ are the height and width of the input frame). We modify the architecture of the $1^{st}$ convolutional layer of VGG-16 such that, it can receive an input of shape $[(3 \times 5) \times H \times W]$, i.e., a set of five stacked optical-flow maps. There are 64 convolutional kernels (filters) of dimension $3 \times 3 \times 3$ in the $1^{st}$ convolutional layer of VGG-16, with the above modification, now we have 64 convolutional kernels of dimension $15 \times 3 \times 3$. We initialise the weights of these new 64 convolutional kernels with the VGG-16 pretrained ImageNet [147] weights for the 1st layer by simply duplicating the weights for 5 times.
7.6.5 Training iterations

We train our model for $320k$ iterations on J-HMDB-21 and $1170k$ iterations on UCF-101-24. The training durations are 3 and 7 GPU days for J-HMDB and UCF-101 respectively.

We follow a same train and test time 3D-ROI sampling strategies as used in Section 6.3.3.
Chapter 8

Conclusions and future directions

[Fabio, please give a pass to this entire chapter.]

In this thesis, we proposed a variety of deep learning based frameworks (cf. Chapter 4, 6 & 7) for action detection in videos. In addition, we addressed the problem of space-time action instance segmentation (cf. Chapter 5). An instance segmentation method can provide more accurate localisation results (than action detection) which are highly beneficial for real-world applications such as self-driving cars.

We presented a novel human action detection approach which addresses in a coherent framework the challenges involved in concurrent multiple human action recognition, spatial localisation and temporal detection, thanks to a novel deep learning strategy for simultaneous detection and classification of region proposals and an improved action tube generation approach (cf. Chapter 4). Our method significantly outperforms the previous state-of-the-art on the most challenging benchmark datasets, for it is capable of handling multiple concurrent action instances and temporally untrimmed videos. Our experimental results demonstrated quantitatively that: (a) a two-stream based deep architecture coupled with our novel test-time fusion technique (for combining appearance and motion cues) significantly improves detection performance; (b) our label smoothing approach ($2^{nd}$ pass of dynamic programming) works gracefully on highly temporally untrimmed videos and improves the detection results; (c) deep feature based supervised region proposals show better recall-to-IoU than unsupervised proposals, (d) using a single-stage detection framework as compared to a multi-stage pipeline helps reducing the computational time and cost by a large margin, and (e) deep feature based action representation is better than shallow representation for action classification task, which further improves the overall action detection results.
Emerging real-world applications (such as self-driving cars) require autonomous systems which can localise action instances at finer pixel-level as compared to coarse bounding-box level detections. The existing action detection approaches are limited to localising actions at bounding-box level and they work in a setting where the videos are temporally trimmed as per the temporal extents of actions and they contain action instances belong to a single action category. We addressed these above drawbacks in the current action detection systems by introducing a novel spatio-temporal action instance segmentation approach which can: (a) delineate action instances in both space and time at pixel-level and classify each instance with both class- and instance-aware labels. (b) perform spatio-temporal action localisation on temporally untrimmed videos containing multiple co-occurring actions belong to same and/or different action categories (cf. Chapter 5). We tested our method on the challenging LIRIS-HARL D2 [14] action detection dataset which contains multiple concurrent actions, with instances belong to same and/or different action class happening at the same time, and where all videos are temporally untrimmed. Our proposed pipeline achieved new benchmark performance which is 14.3 times better than the previous top performer. We were the first to qualitatively demonstrate action instance segmentation results on LIRIS-HARL dataset.

One interesting future direction would be to jointly optimise the human action segmentation, action classification and bounding-box regression objectives under a unified deep learning framework [159] which will then eliminate the need for highly expensive space-time action motion segmentation step [12] and significantly improve the overall detection performance.

The most recent deep learning based action detection approaches [2, 3, 6, 11] follow a frame-level action representation and training. In such methods, rather than learning to classify and regress 3D region proposals, the network learns to classify and regress 2D proposals. Such frame-level solutions are suboptimal and heavily rely on a post-processing step to temporally link frame-level detections to build action tubes. We departed from such current practice in action detection (i.e. frame-level action representation and training) to take a step towards deep network architectures able to classify and regress whole video subsets (i.e. video-level representation and training). In particular, we proposed a novel deep net framework able to regress and classify 3D region proposals spanning two successive video frames, effectively encoding the temporal aspect of actions using just raw RGB values (cf. Chapter 6). We refer this new deep network as “AMTnet” which is end-to-end trainable and can be jointly optimised for action
localisation and classification using a single step of optimisation. At test time
the network predicts ‘micro-tubes’ spanning two frames, which are linked up
into complete action tubes via a new algorithm of our design. Promising results
confirm that AMTnet does indeed outperform the state-of-the-art when relying
purely on appearance.

In Chapter 7 we extended AMTnet deep architecture to improve the ac-
tion representation by integrating the complementary optical flow features in
the same framework. We referred the extended AMTnet as “AMTnet-Flow”.
Unlike AMTnet training, we trained AMTnet-Flow on both short and long dis-
tance frame pairs and at test-time, detection micro-tubes were extracted on long
distance pairs. Our experiments quantitatively demonstrated that: (a) the ac-
tion detection performance is significantly improved by integrating optical flow
based deep features in the same AMTnet framework; (b) the micro-tube extrac-
tion at test-time is made relatively faster with our new bounding-box interpo-
lation algorithm which populates detection bounding-boxes for the intermediate
frames by linear interpolation of box coordinates; (c) our new micro-tube link-
ing algorithm to link relatively longer micro-tubes further reduces the computing
time required for tube generation. We further demonstrated significant improve-
ments in frame-level and video-level mAP over the existing state-of-the-art ap-
proaches \[2, 3, 10, 30, 134\]. Most noticeably, AMTnet-Flow outperforms the top
competitor \[11\] in video-mAP by a large margin of \[6.28\%\] and \[4.79\%\] on higher
IoU thresholds 0.5 and 0.5 : 0.95 respectively on UCF-101-24 action detection
benchmark.

Much work will need to follow. AMTnet-Flow’s combination of high accuracy
and fast detection speed at test time is very promising for real-time applications,
for instance smart car navigation. As the next step we plan to make our tube
generation and labelling algorithm fully incremental and online, by only using
region proposals from independent frames at test time and updating the dy-
namic programming optimisation step at every incoming frame. As the search
space of 3D proposals is twice the dimension of that for 2D proposals, efficient
parallelisation and search are crucial to fully exploit the potential of this ap-
proach. Further down the road we wish to extend the idea of micro-tubes to
longer time intervals, posing severe challenges in terms of efficient regression in
higher-dimensional spaces.
Appendices
Appendix A

Supporting ideas

In this section, we describe some of the supporting ideas at the core of this dissertation. We refer the reader to topics which already have excellent resources available, and elaborate on those for which we feel further clarity is needed in the context of each Chapter.

The 2D connected component and the region proposal generation step is explained in Section A.1. The pruning Selective Search region proposals step is explained in Section A.2.
A.1 Region proposal generation using 2D connected components

[Note for Suman: Put in Appendix, define 2D connected components, and the pseudocode to generate region props.] Let \( S \) represents a subset of pixels in a binary image \( I_b \). Two pixels \( p \) and \( q \) are said to be connected in \( S \) if there exists a path between them consisting entirely of pixels in \( S \). For any pixel \( p \) in \( S \), the set of pixels that are connected to it in \( S \) is called a connected component. We call a set of pixels \( r \) in \( I_b \) a “region” of the image if \( r \) is a connected set. Two regions, \( r_i \) and \( r_j \) are said to be adjacent if their union forms a connected set. Regions that are not adjacent are said to be disjoint. Our region proposal generation algorithm first finds all the disjoint segments \( r \) within the binary segmented image \( I_b \) produced by the human motion segmentation. Each disjoint region is associated with a minimum bounding box \( b_{r_i} \). To get an accurate localization window of the human action, our algorithm takes all the possible combination of these bounding boxes \( b_{r_i} \) to generate \( N \) boxes where:

\[
N = \sum_{k=2}^{n} \frac{n!}{k!(n-k)!},
\]

(A.1)

\( n \) is the number of disjoint region and we take \( k \) between a range 2 to \( n \).

\[1\] We consider 8-adjacency when referring to regions.
A.2 Pruning Selective Search region proposals at test time

In contrast to the work by Gkioxari and Malik [2], we use a smaller motion threshold value to prune Selective Search boxes, to avoid neglecting human activities which exhibit minor body movements exhibited in the LIRIS HARL [14] “typing on keyboard”, “telephone conversation” and “discussion” activities. We apply the motion threshold on the ‘actionness’ scores which we define as:

$$\mu = \frac{\sum_{i \in \mathbf{r}} f_m(i)}{\sum_{j \in \mathbf{I}} f_m(j)}$$  \hspace{1cm} (A.2)

where $f_m(.)$ is the function returning the normalised flow magnitude of each pixel of image $\mathbf{I}$, $i$ and $j$ are the pixel indices for region $\mathbf{r}$ and image $\mathbf{I}$. $\sum_{i \in \mathbf{r}} f_m(i)$ is the ‘actionness’ measure of region $\mathbf{r}$, and the $\sum_{j \in \mathbf{I}} f_m(j)$ is the ‘actionness’ measure of the whole image. $\mu$ is the measure of how much motion or ‘actionness’ is included inside region $\mathbf{r}$.

Even a motion threshold value of 0.003 prunes, on average, 1,000 boxes, thus significantly reducing computation and resources. In this way activities associated with very small body motion such as LIRIS HARL [14]’s “typing on keyboard”, for instance (as opposed to the “running” or “golf” actions of JHMDB21 [?] ) can still be detected using dense optical flow [?].

[Note for Suman: TODO: motion saliency pseudocode is presented. Write a psuedocode from the matvis matlab code and put it in the appendix.]
References


REFERENCES


[21] M. Sapienza, “Recognising and localising human actions.” Available at: https://radar.brookes.ac.uk/radar/items/8c520676-aef0-4b67-a160-b6dd8e6a3e58/1/


REFERENCES


