



# THE UNIVERSITY *of* EDINBURGH

This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

# Localizing spatially and temporally objects and actions in videos

*Vicky Kalogeiton*



Doctor of Philosophy  
Institute of Perception, Action and Behaviour  
School of Informatics  
University of Edinburgh  
2017





# Abstract

The rise of deep learning has facilitated remarkable progress in video understanding. This thesis addresses three important tasks of video understanding: video object detection, joint object and action detection, and spatio-temporal action localization.

Object class detection is one of the most important challenges in computer vision. Object detectors are usually trained on bounding-boxes from still images. Recently, video has been used as an alternative source of data. Yet, training an object detector on one domain (either still images or videos) and testing on the other one results in a significant performance gap compared to training and testing on the same domain. In the first part of this thesis, we examine the reasons behind this performance gap. We define and evaluate several domain shift factors: spatial location accuracy, appearance diversity, image quality, aspect distribution, and object size and camera framing. We examine the impact of these factors by comparing the detection performance before and after cancelling them out. The results show that all five factors affect the performance of the detectors and their combined effect explains the performance gap.

While most existing approaches for detection in videos focus on objects *or* human actions separately, in the second part of this thesis we aim at detecting non-human centric actions, i.e., objects performing actions, such as *cat eating* or *dog jumping*. We introduce an end-to-end multitask objective that jointly learns object-action relationships. We compare it with different training objectives, validate its effectiveness for detecting object-action pairs in videos, and show that both tasks of object and action detection benefit from this joint learning. In experiments on the A2D dataset [Xu et al., 2015], we obtain state-of-the-art results on segmentation of object-action pairs.

In the third part, we are the first to propose an action tubelet detector that leverages the temporal continuity of videos instead of operating at the frame level, as state-of-the-art approaches do. The same way modern detectors rely on anchor boxes, our tubelet detector is based on anchor cuboids by taking as input a sequence of frames and outputting *tubelets*, i.e., sequences of bounding boxes with associated scores. Our tubelet detector outperforms all state of the art on the UCF-Sports [Rodriguez et al., 2008], J-HMDB [Jhuang et al., 2013a], and UCF-101 [Soomro et al., 2012] action localization datasets especially at high overlap thresholds. The improvement in detection performance is explained by both more accurate scores and more precise localization.

**Keywords:** action localization, action recognition, object detection, video analysis, computer vision, deep learning, machine learning

# Acknowledgements

First and foremost, I want to thank my advisors Vittorio Ferrari and Cordelia Schmid. Vitto's passion and Cordelia's vision were the two true driving forces that kept pushing me a step forward. Vitto's ability of instantaneously disassembling an idea and placing into a larger, much wider context is truly admirable. Vitto was a true teacher; I am grateful not just because he taught me how to approach an idea, how to tackle and present it, but most importantly because he taught me how to think. Cordelia's zeal for perfection is truly admirable. Her desire for deep understanding is summarized into one question, that is imprinted in my mind: 'why?'. Whenever we were discussing an idea, a project, or even a result she was determined to unravel all their aspects and discover their true meaning. I am very grateful for her guidance, her support and especially her persistence of pursuing the excellence. Cordelia's deep intuition and vision were my sources of motivation and inspiration.

I would like to thank my jury members, Taku Komura and Tinne Tuytelaars, for accepting to review my thesis and for traveling long distances to attend my viva.

I have been very fortunate to collaborate and be friends with Philippe Weinzaepfel, to whom I am more than grateful. Philippe's feedback, insight, and positive attitude helped me grow as a researcher. The hours we spent discussing and laughing made my last year the most enjoyable period of my PhD.

During my PhD I had the opportunity to interact with people from two great groups, the CALVIN and THOTH (LEAR) groups. My gratitudes to all CALVINees: Holger Caesar, Abel Gonzalez-Garcia, Paul Henderson, Davide Modolo, Dim P. Papadopoulos, Anestis Papazoglou, Luca Del Pero, Jasper Uijlings, Alexander Vezhnets, Michele Volpi for the interesting discussions and fun nights out. Also, from the THOTH group I want to thank Alberto Bietti, Guilhem Cheron, Nicolas Chesneau, Vasilis Choutas, Mikita Dvornik, Maha Elbayad, Albert Gordo, Yang Hua, Hongzhou Lin, Pauline Luc, Thomas Lucas, Xavier Martin, Dan Oneata, Mattis Paulin, Marco Pedersoli, Xiaojiang Peng, Federico Pierucci, Jerome Revaud, Gregory Rogez, Shreyas Saxena, Konstantin Shmelkov, Vladyslav Sydorov, Valentin Thomas, Pavel Tokmakov, Philippe Weinzaepfel, and Daan Wymen. My special thanks to my officemates not only for the fruitful discussions and support but also for making my experience at THOTH so pleasant. Thank you to Heather Low, Magda Nowak, Stephanie Smith, and especially to Nathalie Gillot for helping me with all administrative tasks.

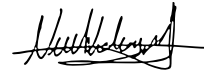
There are so many other people that made my everyday life so much better all these years and were my constant source of energy: Nikos Andrikos, Holger Caesar, Botond

Cseke, Juan Fumero, Stavros Gerakaris, Abel Gonzalez-Garcia, Ioanna Labraki, Maria Leousi, Kostas Nizamis, Dan Oneata, Dim Papadopoulos, John Patlakas, Quentin Pleple, Shreyas Saxena, Aurela Shehu, Konstantin Shmelkov, Panagiotis Theologou, Vagia Tsiminaki, Philippe Weinzaepfel, and Daan Wynen.

Finally, I am more than grateful to my family (my parents and my sister) for their unconditional support and patience through the PhD and for the many lessons they taught me all these years.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.



*(Vicky Kalogeiton)*

# Table of Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                                       | <b>1</b>  |
| 1.1      | Tasks . . . . .   | 3         |
| 1.2      | Challenges on video understanding . . . . .               | 4         |
| 1.3      | Context . . . . .   | 7         |
| 1.4      | Contributions . . . . .                                   | 9         |
| <b>2</b> | <b>Related work</b>                                       | <b>13</b> |
| 2.1      | Deep learning in computer vision . . . . .                | 15        |
| 2.1.1    | Convolutional Neural Networks . . . . .                   | 15        |
| 2.1.2    | Common CNN architectures . . . . .                        | 18        |
| 2.1.3    | AlexNet . . . . .   | 20        |
| 2.1.4    | VGGNet . . . . .  | 21        |
| 2.2      | Object detectors . . . . .                                | 22        |
| 2.2.1    | Deformable Part-based Model . . . . .                     | 24        |
| 2.2.2    | Region-based Convolutional Neural Network . . . . .       | 26        |
| 2.2.3    | Fast R-CNN . . . . .                                      | 28        |
| 2.2.4    | Faster R-CNN . . . . .                                    | 30        |
| 2.2.5    | Single Shot MultiBox Detector . . . . .                   | 32        |
| 2.3      | Domain adaptation . . . . .                               | 34        |
| 2.3.1    | Adapting across different domains . . . . .               | 35        |
| 2.3.2    | Deep domain adaption . . . . .                            | 38        |
| 2.3.3    | Common architectures for deep domain adaption . . . . .   | 40        |
| 2.3.4    | Heterogeneous domain adaptation . . . . .                 | 43        |
| 2.4      | Action localization . . . . .                             | 44        |
| 2.4.1    | Temporal action localization . . . . .                    | 45        |
| 2.4.2    | Spatio-temporal action localization . . . . .             | 46        |
| 2.4.3    | Spatial localization for video object detection . . . . . | 50        |

|          |   |           |
|----------|---|-----------|
| 2.4.4    | Action localization datasets . . . . .  | 52        |
| <b>3</b> | <b>Analyzing domain shift factors between images and videos for object detection</b>          | <b>55</b> |
| 3.1      | Overview of our approach . . . . .  | 57        |
| 3.2      | Related work . . . . .  | 59        |
| 3.2.1    | Adapting object detectors from videos to images . . . . .                                     | 59        |
| 3.3      | Datasets and protocol . . . . .   | 62        |
| 3.3.1    | First dataset pair . . . . .  | 62        |
| 3.3.2    | Second dataset pair . . . . .   | 64        |
| 3.3.3    | Protocol . . . . .  | 64        |
| 3.4      | Analysis of domain shift factors and experimental results . . . . .                           | 66        |
| 3.4.1    | Spatial location accuracy . . . . .   | 67        |
| 3.4.2    | Appearance diversity . . . . .  | 71        |
| 3.4.3    | Image quality . . . . .   | 73        |
| 3.4.4    | Aspect distribution . . . . .   | 76        |
| 3.4.5    | Other factors . . . . .   | 80        |
| 3.4.6    | Experiments on the ILSVRC 2015 dataset pair . . . . .   | 81        |
| 3.5      | Conclusions . . . . .   | 83        |
| <b>4</b> | <b>Object and action detection in videos</b>  | <b>85</b> |
| 4.1      | Overview . . . . .  | 87        |
| 4.2      | Related Work . . . . .  | 89        |
| 4.3      | End-to-end multitask network architecture for joint learning of objects and actions . . . . . | 91        |
| 4.3.1    | End-to-end network architecture . . . . .   | 91        |
| 4.3.2    | Joint learning of objects and actions . . . . .   | 93        |
| 4.4      | Experimental results for multitask learning . . . . .   | 95        |
| 4.4.1    | Joint detection of objects and actions in videos . . . . .                                    | 95        |
| 4.4.2    | Zero-shot learning of actions . . . . .   | 99        |
| 4.4.3    | Object-action segmentation . . . . .  | 100       |
| 4.4.4    | Relationship detection of objects and actions . . . . .                                       | 103       |
| 4.5      | Exploring action adaptation techniques for action localization . . . . .                      | 106       |
| 4.5.1    | Action adaptation . . . . .   | 107       |
| 4.5.2    | Experimental results on action adaptation . . . . .   | 110       |
| 4.5.3    | Action adversarial adaptation . . . . .   | 113       |

|          |   |            |
|----------|---|------------|
| 4.5.4    | Experimental results on action adversarial adaptation . . . . .             | 115        |
| 4.6      | Conclusions . . . . .   | 117        |
| <b>5</b> | <b>Action Tubelet Detector for Spatio-Temporal Action Localization</b>      | <b>119</b> |
| 5.1      | Overview . . . . .  | 120        |
| 5.2      | Related work . . . . .  | 122        |
| 5.3      | Action tubelet detector . . . . .   | 125        |
| 5.3.1    | ACT-detector . . . . .  | 125        |
| 5.3.2    | Two stream ACT-detector . . . . .   | 129        |
| 5.3.3    | From action tubelets to spatio-temporal tubes . . . . .                     | 129        |
| 5.4      | Experimental results . . . . .  | 131        |
| 5.4.1    | Datasets and metrics . . . . .  | 131        |
| 5.4.2    | Implementation details . . . . .  | 132        |
| 5.4.3    | Validation of anchor cuboids . . . . .                                      | 133        |
| 5.4.4    | Tubelet modality . . . . .  | 135        |
| 5.4.5    | Tubelet length . . . . .  | 135        |
| 5.4.6    | Error breakdown analysis . . . . .  | 138        |
| 5.4.7    | Handling moving actors . . . . .  | 139        |
| 5.4.8    | Comparison to other linking methods . . . . .                               | 140        |
| 5.4.9    | Comparison to the state of the art . . . . .                                | 141        |
| 5.5      | Conclusions . . . . .   | 144        |
| <b>6</b> | <b>Conclusions</b>  | <b>147</b> |
| 6.1      | Summary of contributions . . . . .  | 147        |
| 6.2      | Future research and perspectives . . . . .                                  | 150        |
|          | <b>Publications</b>   | <b>155</b> |
| <b>A</b> | <b>YouTube-Objects dataset v2.0</b>   | <b>157</b> |
| <b>B</b> | <b>Additional experimental results of domain shift factors</b>              | <b>159</b> |
| <b>C</b> | <b>Mining video training data</b>   | <b>165</b> |
| <b>D</b> | <b>Additional experimental results on joint object and action detection</b> | <b>169</b> |
| <b>E</b> | <b>ACT-detector: additional experimental results</b>                        | <b>175</b> |





# List of Figures

|     |  |   |
|-----|--|---|
| 1.1 | <i>Examples of two pictures taken from a Google car: (first row) what the self-driving car sees, (second row) photos taken from three different sides. Image source [Urmson, 2015]. . . . .</i>  | 2 |
| 1.2 | <i>Examples of object classes in video object detection. Frames from the YouTube-Objects dataset [Kalogeiton et al., 2016; Prest et al., 2012a].</i>   | 3 |
| 1.3 | <i>Objects performing actions from the A2D dataset [Xu et al., 2015]. . .</i>  | 3 |
| 1.4 | <i>Spatio-temporal action localization. Examples from the UCF-Sports dataset [Rodriguez et al., 2008]. . . . .</i>   | 4 |
| 1.5 | <i>Example of intra-class variations, where three instances of the same object class car have very different appearance: different car models, shapes, colors, etc. Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and ILSVRC VID 2015 [Russakovsky et al., 2015a] datasets. . . . .</i> | 5 |
| 1.6 | <i>Example of (first row) different viewpoints, and (second row) occlusions. Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and UCF-101 [Soomro et al., 2012] datasets. . .</i>  | 5 |
| 1.7 | <i>Examples of differences in the capturing style: (first row) panning, and (second row) tilting. Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and UCF-101 [Soomro et al., 2012] datasets. . . . .</i>   | 6 |
| 1.8 | <i>Examples of differences in the capturing style: (first row) panning, and (second row) tilting. Examples from the J-HMDB [Jhuang et al., 2013a] dataset. . . . .</i>   | 6 |
| 1.9 | <i>Examples of difficulties in action localization, due to different actions appearing in the videos. Examples from the UCF-Sports [Rodriguez et al., 2008] dataset. . . . .</i>   | 7 |

|      |  |    |
|------|--|----|
| 1.10 | <i>Chapter 3: Example of image quality difference between video frame (left) and images (right). Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and PASCAL VOC 2007 [Everingham et al., 2007] datasets.</i>  | 9  |
| 1.11 | <i>Chapter 4: Our two-stream end-to-end network for joint detection of objects and actions.</i>  | 10 |
| 1.12 | <i>Chapter 5: Our ACT-detector takes as input sequences of frames and predicts tubelets.</i>   | 10 |
| 2.1  | <i>Example of convolution operation. Given an image (input) and a filter (kernel), the convolution operation computes the weighted sum of a pixel and its neighbors (dark blue and blue) and produces the output pixel value (red in output). Image source <sup>1</sup></i>  | 16 |
| 2.2  | <i>An illustration of the AlexNet architecture. It shows the split into two parts, each one operating on a different GPU. Image source [Krizhevsky et al., 2012].</i>  | 20 |
| 2.3  | <i>An illustration of the VGG16 architecture. Image source Noh et al. [2015]</i>   | 21 |
| 2.4  | <i>An illustration of DPM: (left) image pyramid, (middle) HOG feature pyramid, and (right) root filter, part filters and spatial models for the person class. The visualization of the spatial models reflects the cost of placing the center of a part at different locations relative to the root. Image source [Girshick et al., 2012a].</i>  | 24 |
| 2.5  | <i>An illustration of the R-CNN detector. For a test image, (1) R-CNN extracts class-generic region proposals, (2) computes their CNN features, and (3) classifies each proposal using class-specific linear SVMs (4). Image source [Girshick et al., 2014a].</i>  | 26 |
| 2.6  | <i>An illustration of the Fast R-CNN model: the input image and multiple RoIs are fed into a fully-convolutional network. Each RoI is pooled and mapped to a feature vector. The network branches into two layers, a softmax layer, that outputs softmax probabilities, and a bounding box regression layer, that predicts the four coordinates of the bounding boxes. Image source [Girshick, 2015a].</i> | 28 |

|      |   |    |
|------|---|----|
| 2.7  | <i>An illustration of the Faster R-CNN detector. One network with four losses. The first two losses belong to the RPN. Image source [Ren et al., 2015a].</i>  | 30 |
| 2.8  | <i>An illustration of SSD. It adds several conv layers to the end of a base network. The extra conv layers predict offsets to anchor boxes of different scales and aspect ratios and their associated confidences. Image source [Liu et al., 2016a].</i>  | 32 |
| 2.9  | <i>Example of domain adaptation, where goal is to learn a discriminative mapping of target images to the source feature space (target encoder) by fooling a domain discriminator, which tries to distinguish the encoded target images from source examples. Image source [Tzeng et al., 2017].</i>       | 34 |
| 2.10 | <i>CNN architecture for domain adaptation with confusion loss and data classifier. Image source [Tzeng et al., 2015].</i>   | 41 |
| 2.11 | <i>An unsupervised domain adaptation architecture. It includes a deep feature extractor (green), a deep label predictor (blue), and a domain classifier (red). Image source [Ganin and Lempitsky, 2015].</i>  | 42 |
| 2.12 | <i>Example of a modern two-stream CNN architecture for predicting action labels. Image source [Weinzaepfel et al., 2015].</i>   | 48 |
| 2.13 | <i>From top to bottom, example frames from the UCF-Sports, J-HMDB, and UCF-101 action localization datasets.</i>  | 52 |
| 3.1  | <i>Our pipeline for examining the impact of domain shift factors between videos and images for object detection. First, we train two object detectors (DPM and R-CNN) on one source domain (videos or images). Then, we test on both domains and examine the difference in the detection performance.</i> | 57 |
| 3.2  | <i>Example YTO frames with ground-truth bounding-boxes.</i>   | 62 |
| 3.3  | <i>Example bounding-boxes produced by PRE [Prest et al., 2012a] (red), FVS [Papazoglou and Ferrari, 2013] (blue), and ground-truth annotations (green).</i>   | 68 |
| 3.4  | <i>mAP results: impact of the domain shift factors when training on VOC and YTO (a) the DPM detector and (b) the R-CNN detector for the testVOC.</i>  | 69 |

|      |   |    |
|------|---|----|
| 3.5  | <i>mAP results: impact of the domain shift factors when training on VOC and YTO (a) the DPM detector and (b) the R-CNN detector for the testYTO.</i>  | 70 |
| 3.6  | <i>(top row) YTO dataset: Frames in the same shot that contain near identical samples of an object. (bottom row) VOC dataset: Example of near identical samples in the same image.</i>  | 71 |
| 3.7  | <i>Four example groups of near-identical samples in trainYTO. We display a subset of the frames for each group.</i>   | 72 |
| 3.8  | <i>Video frame, VOC training image, Gaussian and motion blurred VOC training images.</i>  | 74 |
| 3.9  | <i>2D visualization of the trainYTO Unique Samples (red) and train-VOC Motion Blurred Unique Samples (green) for the ‘horse’ class in R-CNN feature space. Circles indicate the samples selected by our equalization technique of Section 3.4.4: Equalization.</i>  | 77 |
| 3.10 | <i>Evolution of <math>d_{KL}</math> between the image and video training sets as for each training set our algorithm adds more and more sample pairs. The two distributions are very similar at the beginning and start to diverge later. The plot corresponds to the object horse, where at the <math>\epsilon = 0.1</math> threshold 70 samples from each set are selected. This number is driven by <math>\epsilon</math> and changes from class to class.</i> | 78 |
| 3.11 | <i>Aspects common to both VOC (green) and YTO (red). Both datasets contain samples with same aspects, such as cat faces, dog sitting, or horse running.</i>   | 79 |
| 3.12 | <i>Different aspects between VOC (green) and YTO (red). Top row: aspects occurring only in VOC. Chicken and birds flying are common in VOC, but do not appear in YTO. Bottom row: aspects occurring only in YTO. YouTube users often film their own pets doing funny things, such as jumping, rolling and going on a skateboard.</i>  | 80 |
| 3.13 | <i>mAP results: impact of the domain shift factors when training on ILSVRC IMG and VID the R-CNN detector and testing on (a) ILSVRC IMG, and (b) ILSVRC VID.</i>  | 82 |
| 4.1  | <i>Detection examples of different object-action pairs for the videos of the A2D dataset <a href="#">Xu et al. [2015]</a>.</i>  | 87 |

|     |  |     |
|-----|--|-----|
| 4.2 | <i>Overview of our end-to-end multitask network architecture for joint object-action detection in videos. Blue color represents convolutional layers while green represents fully connected layers. The end-to-end training is done by concatenating the fully connected layers from both streams. . . . .</i>   | 91  |
| 4.3 | <i>Illustration of the three different ways we consider for jointly learning objects and actions. The blue nodes represent objects and the red ones action classes, while the yellow ones represent the background class.</i>  | 93  |
| 4.4 | <i>Overview of our setup for extending our object-action detections to semantic segmentations using segmentation proposals from [Grundmann et al., 2010; Pinheiro et al., 2016]. For each bounding-box detection, e.g. dog running we find the segmentation proposals that overlaps mostly with the detection and use it as the final segmentation.</i>                  | 101 |
| 4.5 | <i>Examples of semantic segmentation with (from left to right): the frame, the ground-truth and the segmentation output obtained when combining our approach with proposals from SharpMask Pinheiro et al. [2016]. The colors of the segmentations represent an object-action pair. Note that we do not use any object-action segmentation at training time. . . . .</i> | 103 |
| 4.6 | <i>Qualitative object-action relationship results on the VRD dataset. The yellow color depicts our correct boxes with their green label, while the red color represents missed interactions until R@100. . . . .</i>   | 105 |
| 4.7 | <i>Illustration of the five different early fusion strategies: fusing at the (a) conv1, (b) conv5, (c) fc6, and (d) softmax layer. . . . .</i>   | 109 |
| 4.8 | <i>Illustration of the adversarial fusion network. The two streams are fused in the fc6 and at the conv5, where two losses are computed: <math>p_D</math> for the domain discriminator and <math>p_M</math> for the domain confusion loss.</i>   | 114 |
| 5.1 | <i>Understanding an action from a single frame can be ambiguous, e.g. sitting down or standing up; the action becomes clear when looking at a sequence of frames. . . . .</i>  | 121 |

|      |   |     |
|------|---|-----|
| 5.2  | <i>Overview of our ACT-detector. Given a sequence of frames, we extract convolutional features with weights shared between frames. We stack the features from subsequent frames to predict scores and regress coordinates for the anchor cuboids (middle figure, blue color). Depending on the size of the anchors, the features come from different convolutional layers (left figure, color coded: yellow, red, purple, green). As output, we obtain tubelets (right figure, yellow color).</i> | 122 |
| 5.3  | <i>Our ACT-detector: construction of the anchor cuboids from different convolutional layers, such as the blue color in the figure. (right) The anchor cuboid over time.</i>   | 126 |
| 5.4  | <i>Example of regressed tubelet (yellow) from a given cuboid (cyan) in our proposed ACT-detector. Note the accurate localization of the tubelet, despite the fact that the aspect ratio of the cuboid is changing heavily across time.</i>  | 128 |
| 5.5  | <i>Toy example of constructing tubes (orange) from tubelets (blue). Tubelets overlap over <math>K</math> frames. At each frame we select the tubelet with the highest score and high overlap with the last tubelet of the link. The score and bounding boxes of a tube are the average of the scores and bounding box coordinates of its tubelets.</i>  | 130 |
| 5.6  | <i>Motion overlap: Mean motion overlap between a box in a ground-truth tube and its box <math>n</math> frames later for varying <math>n</math>.</i>   | 133 |
| 5.7  | <i>(a-c) Recall of the anchor cuboids for various IoU thresholds on the training set of three action localization datasets. The numbers in parenthesis indicate the recall at <math>\text{IoU} = 0.5</math>.</i>  | 134 |
| 5.8  | <i>Recall of the anchor cuboids at <math>\text{IoU} = 0.5</math> on the training set of three action localization datasets.</i>   | 134 |
| 5.9  | <i>Frame-mAP of our ACT-detector on the three datasets when varying <math>K</math> for RGB data (blue line), flow (red line), union and late fusion of RGB + flow data (black and green lines, resp.).</i>  | 135 |
| 5.10 | <i>MABO (top) and classification accuracy (bottom) of our ACT-detector on the three datasets when varying <math>K</math>. For J-HMDB and UCF-101 we report results on the first split.</i>  | 136 |

|      |  |     |
|------|--|-----|
| 5.11 | <i>Examples when comparing per-frame (<math>K=1</math>) and tubelet detections (<math>K=6</math>). The yellow color represents the detections and their scores for the classes shown, the red color highlights errors either due to missed detections (first column) or wrong labeling (third column) while the green color corresponds to correct labels. Our ACT-detector outputs one class label with one score per tubelet, we thus display it once. . . . .</i> | 137 |
| 5.12 | <i>Error analysis of our ACT-detector for sequence length <math>K = 1</math> and <math>K = 6</math> on three action localization datasets. We show frame-mAP and different sources of error, see Section 5.4.6 for details. For J-HMDB and UCF-101 we report results on the first split. . . . .</i>   | 138 |
| 5.13 | <i>video-MABO of our tubelet detections with different linking methods. We employ the methods of Peng and Schmid [2016] ('method1'), and of Singh et al. [2017] ('method2'). . . . .</i>   | 140 |
| 5.14 | <i>Examples of our generated tubes (yellow) with the ground-truth tubes (green) for videos of the UCF-101 [Soomro et al., 2012] dataset. . . . .</i>   | 143 |
| 5.15 | <i>Failure (red) and correct cases (yellow) of our generated tubes with the ground-truth tubes (green) for videos of the UCF-101 [Soomro et al., 2012] dataset. . . . .</i>  | 144 |
| 6.1  | <i>Example of pipeline that models interactions between humans and objects. Image from [Prest et al., 2013]. . . . .</i>   | 150 |
| 6.2  | <i>Three-stream network for action classification: RGB, optical flow and pose. Image from [Zolfaghari et al., 2017]. . . . .</i>   | 151 |
| 6.3  | <i>Example of the household activities and sub-activities tree. Image from ActivityNet [Heilbron et al., 2015] for activity detection. . . . .</i>   | 153 |
| C.1  | <i>2D visualization of the training and test sets (a) before and (b) after selecting VIRAT samples. We select as many VIRAT samples as there are in the first 10 NNs of each missed detection and of each correct detection (testVOC). . . . .</i>   | 166 |
| C.2  | <i>% mAP for the class car of the R-CNN and fast R-CNN detectors for different training sets. Each training set contains the 1644 VOC training samples and some VIRAT samples. . . . .</i>   | 167 |
| C.3  | <i>% statistics of the detections before and after adding the VIRAT training samples for the class car. . . . .</i>  | 168 |





# List of Tables

|     |  |     |
|-----|--|-----|
| 2.1 | <i>Overview of the action localization datasets used in our experiments.</i>   | 53  |
| 3.1 | Number of object samples in the training and test sets for image (VOC) and video (YTO) domains. . . . .  | 66  |
| 3.2 | Appearance diversity equalization. Statistics of the groups: number of groups, ratio: number of groups / number of ground-truth samples and number of equalized unique samples. . . . .  | 75  |
| 4.1 | <i>Comparison of different losses for object-action learning. We give the number of parameters in the classification layers from the VRD dataset [Lu et al., 2016] where <math> O  = 100,  \mathcal{A}  = 140,  \mathcal{V}  = 13344</math> (Section 4.4.4).</i> . . . . . | 92  |
| 4.2 | <i>Overview of the video datasets used in our experiments.</i> . . . . .   | 96  |
| 4.3 | <i>Impact of end-to-end training: mAP for object detection of different training scenarios on the A2D, YTO and VID datasets.</i> . . . . .   | 97  |
| 4.4 | <i>mAP of six different models when training with objects (first row), actions (second row), when multiplying their scores (third row) or when jointly training with objects and actions (last three rows) on A2D.</i> . . . .   | 98  |
| 4.5 | <i>Evaluation of zero-shot learning for object-action pairs on A2D. For each object, we report the AP when excluding all actions of this object at training. The numbers in parenthesis indicate the AP when training with all object-action pairs.</i> . . . . .          | 100 |
| 4.6 | <i>Comparison to the state of the art for object, action and object-action segmentation on A2D using class-average pixel accuracy (ave), global pixel accuracy (glo) and mean Intersection over Union (mIoU) metrics.</i> . . . . .  | 102 |

|      |   |     |
|------|---|-----|
| 4.7  | <i>Comparison of our multitask model to baselines and to the state-of-the-art visual relationships results on the VRD dataset for phrases and relationship detection. We report <math>R@100</math> and <math>R@50</math> for methods using only visual cue (V) or also language and frequency priors (V+L+F).</i> | 104 |
| 4.8  | <i>Comparison of our multitask model to the state-of-the-art methods for zero-shot detection of visual relationships on the VRD dataset. We report <math>R@100</math> and <math>R@50</math> for methods using only visual cue (V) or also language and frequency priors (V+L+F).</i>                              | 106 |
| 4.9  | <i>Frame-mAP for action detection of different training scenarios on the UCF-Sports, J-HMDB and UCF-101 datasets. For more details see Section 4.5.2.</i>   | 111 |
| 4.10 | <i>Frame-mAP for action localization of different training scenarios with multi-scale testing on the UCF-Sports, J-HMDB and UCF-101 datasets. The test scales we use are [480, 600, 800].</i>   | 112 |
| 4.11 | <i>Frame-mAP for action localization for the <math>ef_6</math> fusion scenario with multi-scale training and testing on the UCF-Sports, J-HMDB and UCF-101 datasets. Both training and test scales are [480, 600, 800].</i>   | 113 |
| 4.12 | <i>Frame-mAP for action localization for the <math>\tilde{ef}_6</math> adversarial network with one and multiple training scales on the UCF-Sports, J-HMDB and UCF-101 datasets.</i>  | 116 |
| 5.1  | <i>Frame-mAP for slow, medium and fast moving actors.</i>   | 139 |
| 5.2  | <i>Comparison of different linking methods. For our tubelet detections we computed the video-mAP with various state-of-the-art linking strategies. For J-HMDB we report results averaged over all splits, and for UCF-101 we report results on the first split.</i>   | 140 |
| 5.3  | <i>Comparison of frame-mAP to the state of the art. For <a href="#">Peng and Schmid [2016]</a>, we report the results with and without their multi-region (+MR) approach. For J-HMDB we report results averaged over all splits, and for UCF-101 we report results on the first split.</i>                        | 142 |

|     |   |     |
|-----|---|-----|
| 5.4 | <i>Comparison of video-mAP to the state of the art at various detection thresholds. The columns 0.5:0.95 correspond to the average video-mAP for thresholds in this range. For Peng and Schmid [2016], we report the results with and without their multi-region (+MR) approach. For J-HMDB we report results averaged over all splits, and for UCF-101 we report results on the first split.</i> | 145 |
| A.1 | <i>Statistics of videos and frames in the YTO v2.0.</i>   | 158 |
| A.2 | <i>Statistics of annotations in the YTO v2.0.</i>   | 158 |
| B.1 | <i>Number of training object samples for the VOC-YTO dataset pair before cancelling out each factor.</i>  | 160 |
| B.2 | <i>Per-class mAP results of the DPM detector when testing on VOC.</i>   | 160 |
| B.3 | <i>Per-class mAP results of the R-CNN detector when testing on VOC.</i>   | 161 |
| B.4 | <i>Per-class mAP results of the DPM detector when testing on YTO.</i>   | 161 |
| B.5 | <i>Per-class mAP results of the R-CNN detector when testing on YTO.</i>   | 162 |
| B.6 | <i>Number of training object samples for the ILSVRC VID-IMG dataset pair before cancelling out each factor.</i>   | 162 |
| B.7 | <i>Per-class mAP results of the R-CNN detector when testing on ILSVRC IMG.</i>  | 163 |
| B.8 | <i>Per-class mAP results of the R-CNN detector when testing on ILSVRC VID.</i>  | 163 |
| D.1 | <i>Per-class mAP results on the objects of the A2D dataset while examining the end-to-end training.</i>   | 170 |
| D.2 | <i>Per-class mAP results on the objects of the YTO dataset while examining the end-to-end training.</i>   | 170 |
| D.3 | <i>Per-class mAP results on the objects of the ILSVRC 2015 VID dataset while examining the end-to-end training.</i>   | 171 |
| D.4 | <i>Per-class mAP results for different architectures when testing only on the objects of the A2D dataset.</i>   | 172 |
| D.5 | <i>Per-class mAP results for different architectures when testing only on the actions of the A2D dataset.</i>   | 172 |
| D.6 | <i>Per-class mAP results for different architectures when testing both on the objects and on their actions for the A2D dataset.</i>   | 173 |

|     |   |     |
|-----|---|-----|
| D.7 | <i>Comparison of per-class segmentation results (ave and glo) when using object-action labels on the A2D dataset. . . . .</i> | 174 |
| E.1 | <i>Per-class results for frame-related evaluations on UCF-Sports. . . . .</i>   | 176 |
| E.2 | <i>Per-class results for frame-related evaluations on J-HMDB. . . . .</i>   | 177 |
| E.3 | <i>Per-class results for frame-related evaluations on UCF-101. . . . .</i>  | 178 |
| E.4 | <i>Per-class results for video-related evaluations on UCF-Sports. . . . .</i>   | 179 |
| E.5 | <i>Per-class results for video-related evaluations on J-HMDB. . . . .</i>   | 180 |
| E.6 | <i>Per-class results for video-related evaluations on UCF-101. . . . .</i>  | 181 |
| E.7 | <i>Frame-mAP and video-mAP per-class results J-HMDB splits 2 and 3. . . . .</i>   | 182 |

# Chapter 1

## Introduction

### Contents

---

|     |   |   |
|-----|---|---|
| 1.1 | Tasks . . . . .                             | 3 |
| 1.2 | Challenges on video understanding . . . . . | 4 |
| 1.3 | Context . . . . .                           | 7 |
| 1.4 | Contributions . . . . .                     | 9 |

---

**F**igure 1.1 shows a crossing from three different viewpoints and their combination into a fine 3D representation. The car stops at a red traffic light, identifying and localizing each moving vehicle. It also spots a bicycle coming from the left side; it localizes the bicycle and predicts its trajectory. When the traffic light turns green, it will cross the bicycle’s trajectory on the street—attention should be paid.

This is a real-time modern computer vision system from a Google autonomous car. The car is able to analyze its surrounding in real time, accurately detect humans and objects, and anticipate their movements. There are prototype cars that drive by themselves, but without prototype vision they cannot tell the difference between a moving truck and a racing ambulance with flashing lights that needs to overtake all other vehicles.

Security cameras are everywhere; yet they cannot recognize a girl being mugged in a bus. Drones can fly over vast areas, but they do not have enough vision to understand if somebody is swimming or drowning. Autonomous robots can take care of our homes, but they can not tell the difference between somebody sleeping or fainting. Surgical robots assist doctors and nurses; yet a fifteen-hours surgery still depends on the tireless eyes of surgeons.

These are some examples of the limitations that vision technology systems face

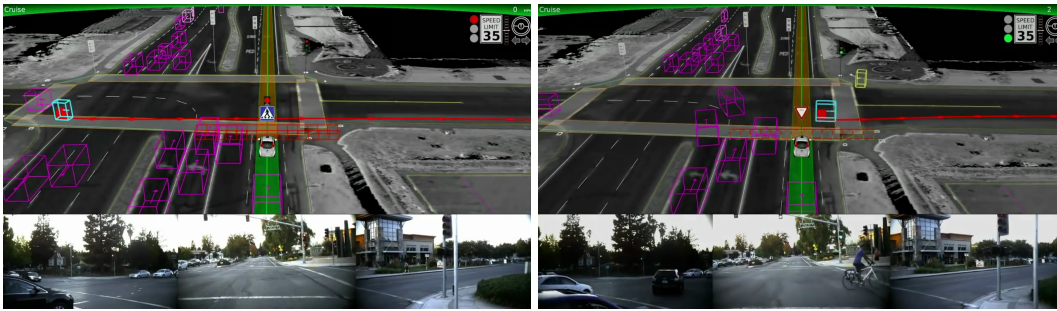


Figure 1.1: Examples of two pictures taken from a Google car: (first row) what the self-driving car sees, (second row) photos taken from three different sides. Image source [Urmson, 2015].

as of today. Nevertheless, the efforts of the computer vision community have led to impressive results on several tasks, such as face recognition and object detection [Girshick et al., 2014a; Liu et al., 2016a; Phillips and O’toole, 2014; Taigman et al., 2014]. For instance, Facebook automatically tags known people in any newly uploaded photo, and Google Photos groups photos by people, objects, or scenes. Video understanding, however, entails the development of tools that effectively analyze the tremendous amount of data that video content provides.

The core of this thesis is video understanding—a fundamental problem in computer vision. Our goal is to first understand the differences between videos and images, then to jointly detect objects and actions in videos, and finally to automatically localize human actions spatially and temporally. The strongest case for the importance of video understanding comes from the wide-spread adoption of videos. We are using applications based on video recognition on a daily basis: almost all cameras in our mobile phones perform face recognition or even adapt the lighting of the videos we take. The Kinect console from Microsoft performs real-time pose estimation allowing its users to control their movements; video-surveillance cameras monitor activities and predict actions. Google Glass records all the daily activities, resulting in an enormous amount of data. The most characteristic example consists in indexing and managing large video collections on video-sharing websites, such as YouTube, Vimeo, etc. For instance, in 2017, almost *five billion* videos<sup>1</sup> are watched on YouTube every single day, and almost *300 hours* of videos<sup>2</sup> are uploaded every minute!

<sup>1</sup><http://www.statisticbrain.com/youtube-statistics/>

<sup>2</sup><https://web.archive.org/web/20150429201745/https://www.youtube.com/yt/press/statistics.html>

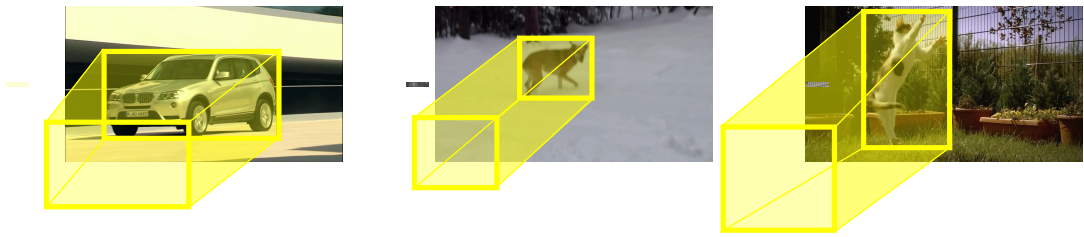


Figure 1.2: Examples of object classes in video object detection. Frames from the YouTube-Objects dataset [Kalogeiton et al., 2016; Prest et al., 2012a].

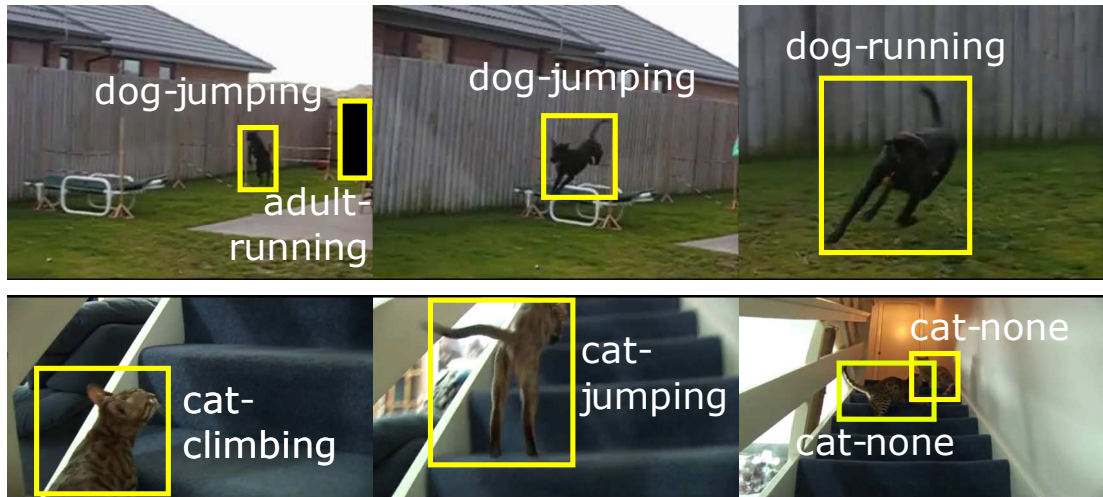


Figure 1.3: Objects performing actions from the A2D dataset [Xu et al., 2015].

## 1.1 Tasks

This dissertation addresses three important problems of video understanding: video object detection, joint object and action detection in videos, and spatio-temporal action localization. Albeit distinct, these tasks are intrinsically connected: they all require a rich video representation, which is robust to intra-class variations but captures enough information to allow for accurate classification and localization. In this section, we first define each task, and then present the challenges these tasks face.

**Video object detection.** The vision community casts video object detection as detection in static frames, being treated as still images. The goal of object class detection is to recognize objects as belonging to certain classes, and localizing them spatially. The spatial localization is addressed by predicting an axis-aligned rectangle (called bounding box) around each instance of an object class. In object detection in images, the object classes consist of both moving and static objects, such as *table* or *bottle*. In contrast, in video object detection typically the object classes have motion, including





Figure 1.4: *Spatio-temporal action localization. Examples from the UCF-Sports dataset [Rodriguez et al., 2008].*

moving animals, such as *birds* or *horses*, or rigid moving objects, such as *aeroplanes* or *cars*. See Figure 1.2 for examples of objects classes.

**Object and action detection in videos.** Similarly to object detection, object and action detection is the task of naming and localizing both object instances and the actions they perform. The localization takes place spatially, with a bounding box around an object instance. Although equivalent to object detection, the classification of each box requires more advanced analysis, as it consists of predicting a pair of labels—one for the object and one for the action class. Objects typically consist of animals, such as *dog* or *cat*, and of humans, such as *adult* or *baby*. Actions refer to atomic actions, such as *climb*, *jump*, or *run*. Figure 1.3 shows some examples of objects performing actions from the A2D dataset [Xu et al., 2015].

**Spatio-temporal action localization in videos.** Action localization is the task of identifying where an action in a video takes place. There are two types of action localization: temporal localization and spatio-temporal localization. In temporal localization the goal is to find the temporal span of an action, i.e., the beginning and ending frames. Spatio-temporal localization finds where an action occurs in space *and* in time, i.e., temporally finding the beginning and ending frames of the action, and spatially finding the bounding boxes that cover the action. This dissertation addresses human action localization in both space and time, in uncontrolled videos. Some examples of human action localization include *diving*, *kick*, *swing*, see Figure 1.4.

## 1.2 Challenges on video understanding

Here, we briefly present the main challenges of video understanding grouped into two categories: intra-class variation and across class similarities.



Figure 1.5: Example of intra-class variations, where three instances of the same object class car have very different appearance: different car models, shapes, colors, etc. Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and ILSVRC VID 2015 [Russakovsky et al., 2015a] datasets.

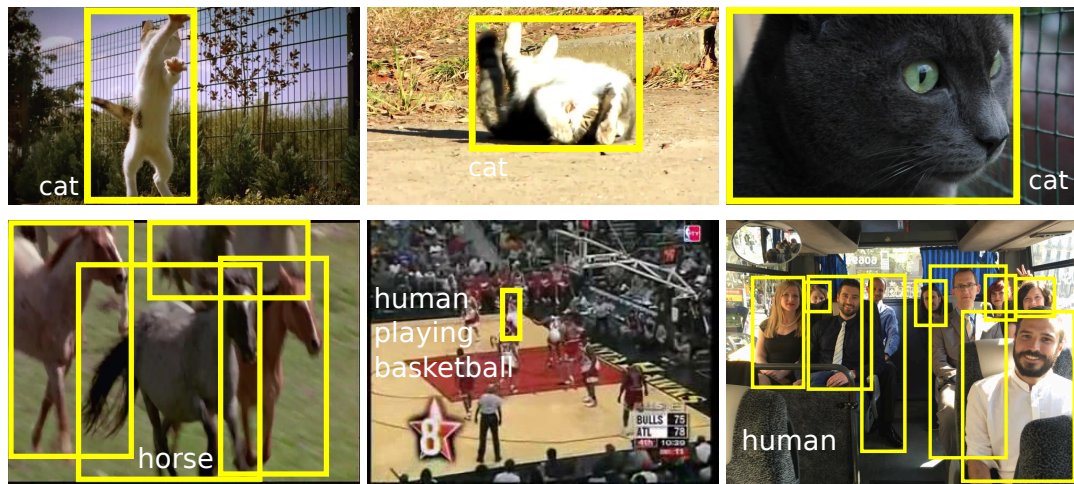


Figure 1.6: Example of (first row) different viewpoints, and (second row) occlusions. Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and UCF-101 [Soomro et al., 2012] datasets.

**Intra-class variation** is a major challenge in video detection. The appearance and motion of instances within a class differ significantly from other instances of the same class, rendering their classification very difficult. Figure 1.5 illustrates some examples of video frames with high intra-class variations.

These differences are grouped into two categories. The first one concerns the object/actor style, such as viewpoints, poses, aspect, occlusion, cluttered background, actor's motion speed, etc. For instance, the first row of Figure 1.6 clearly demonstrates differences in viewpoints and spatial location, whereas the second row shows occlusion examples. The second category includes differences in the capturing style, such as camera framing and lens, resolution, zooming, panning, tilting, etc. Figure 1.7 shows some examples of panning (top) and tilting (bottom) for object and action classes.

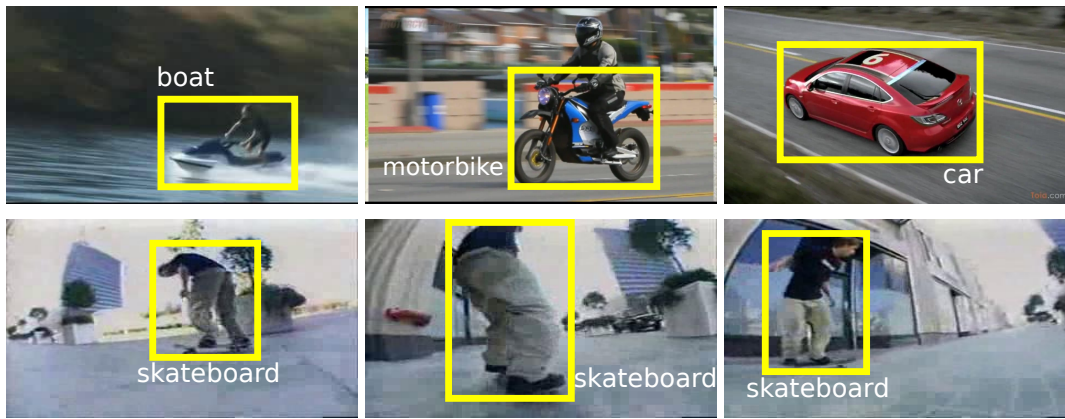


Figure 1.7: Examples of differences in the capturing style: (first row) panning, and (second row) tilting. Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and UCF-101 [Soomro et al., 2012] datasets.



Figure 1.8: Examples of differences in the capturing style: (first row) panning, and (second row) tilting. Examples from the J-HMDB [Jhuang et al., 2013a] dataset.

**Across-class similarities** is another challenge. Objects or actions of different classes share similar characteristics, especially motion-wise. Modern methods typically process videos at the frame level. Nevertheless, processing frames individually is not optimal, as distinguishing actions from a single frame can be ambiguous. Figure 1.8 shows three examples of humans performing different actions, which are indeed indistinguishable: *walking*, *catching*, or *running*. Processing, however, sequences of frames can help distinguishing between actions.

An interesting challenge of across-class similarities refers to the temporal inconsistency, especially of actions. Actors or objects are typically labeled as to perform an action at the video level not at the frame level. This potentially creates several confusions, especially when considering per-frame localizations. Figure 1.9 illustrates such an example: when considering a video labeled with the action *kick*, plenty of its frames depict the girl *running*, i.e., a different action class. For instance, in object detection



Figure 1.9: *Examples of difficulties in action localization, due to different actions appearing in the videos. Examples from the UCF-Sports [Rodriguez et al., 2008] dataset.*

the frames of Figure 1.9 would correctly have different labels, whereas a naive video classification system could predict the action *kick* in this video just by considering the most appearing label. To really understand videos though, an effective temporal localization system should be able to isolate the temporal window that the action occurs and discard the rest. Therefore, the temporal localization dramatically increases the difficulty of the task, especially when considering videos of several minutes with the action lasting only a few seconds.

## 1.3 Context

Video understanding is one of the most challenging research areas in computer vision. Three years old children can make sense of what they see when looking at a video, for example a group of people playing basketball. Yet our most advanced machines and computers still struggle at this task. Understanding this requires not only identifying the type of objects or actors appearing in a video, such as humans and ball, but also localizing them spatially and temporally.

Enabling computers to understand what a scene is about started in the 1970s. The earliest approaches try to segment moving objects in a video [Jain et al., 1977; Martin and Aggarwal, 1977]. Later on, Hogg [1983] analyzes human motion in videos by building a 3D human model based on cylinders. The human is detected by first subtracting the background, then extracting the edges; the method is evaluated by projecting the human model onto the images.

The first action recognition attempts are traced back in the 1990's with 3D volumetric human models [Campbell et al., 1996; Rohr, 1994], which were progressively replaced by simpler models like 2D silhouettes [Brand, 1999; Yamato et al., 1992]. Spatio-temporal action localization though is still a recent field of research, which has been growing swiftly in the past couple of decades. The initial attempts for action lo-



calization are extensions of the sliding window scheme [Cao et al., 2010; Laptev and Pérez, 2007], requiring strong assumptions such as a cuboid shape, i.e., a fixed spatial extent of the actor across frames, or figure-centric models, which leverage human detectors [Klaser et al., 2010] or treat the actor position as a latent variable [Lan et al., 2011]. More recently, several approaches extend methods from object detection in images to build robust action localization systems. For instance, Tian et al. [2013] extend the deformable parts model [Girshick et al., 2012b] to videos, whereas Jain et al. [2014]; Oneata et al. [2014a] extend object proposals [Uijlings et al., 2013] to action proposals.

Joint object-action detection in videos is yet another growing task in computer vision. Several authors tackle this problem as human-object interactions [Filipovych and Ribeiro, 2008, 2011; Gupta et al., 2009; Matikainen et al., 2010; Messing et al., 2009]. For instance, Filipovych and Ribeiro [2008] and Filipovych and Ribeiro [2011] use controlled videos to model human-object interactions based on the trajectory and appearance of spatio-temporal interest points. Gupta et al. [2009] model the action of an object and the action of a human-object pair using hand trajectories to describe how objects are reached and grasped. However, the task of object-action detection consists in detecting objects being the actors of the actions, and it is still in its infancy. Bojanowski et al. [2013] study the case of different entities performing a set of actions, but these entities correspond to names of different actors. Closely related to object-action detection are the works on segmenting object-action from Xu and Corso [2016]; Xu et al. [2015], where they output object-action semantic segmentations.

Interestingly, with a few exceptions [Misra et al., 2015; Pirsiavash et al., 2011], object detection in videos has been disregarded by the vision community. One of the reasons for that is the cost of large datasets, including annotating and handling them. In one of the first works, Ramanan et al. [2006] build animal models for tracking and detection, whereas Ommer et al. [2009] learn detection models as 3D points using structure from motion. Most of the existing works use videos to help detecting objects on images [All et al., 2011; Hartmann et al., 2012; Leistner et al., 2011; Prest et al., 2012a; Tang et al., 2013]. Only during the last few years with the evolution of big data, there are some attempts that exploit the temporal continuity of moving objects for video object detection [Kang et al., 2016b,a, 2017; Zhu et al., 2017b,a]. These works rely on per-frame object detections, and then leverage the motion of objects to refine their spatial localization or improve their classification.

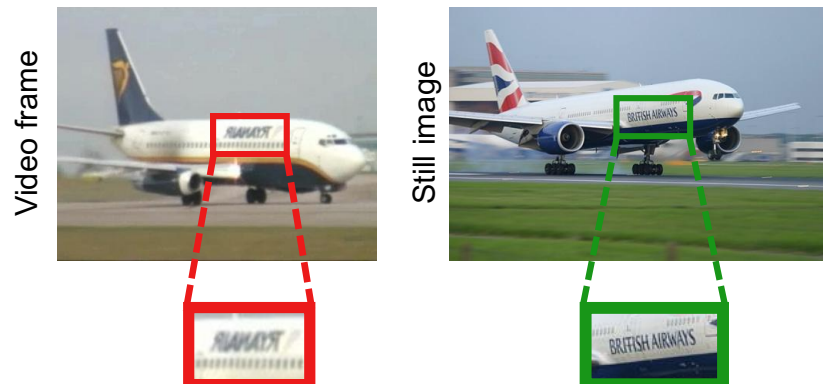


Figure 1.10: Chapter 3: Example of image quality difference between video frame (left) and images (right). Examples from the YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a] and PASCAL VOC 2007 [Everingham et al., 2007] datasets.

## 1.4 Contributions

This thesis makes the following three main contributions:

- We explore the differences between still images and video frames for training and testing an object detector. We consider five domain shift factors that make videos different from still images: spatial location accuracy, appearance diversity, image quality, aspect distribution, and camera framing. Following a systematic protocol, we analyze them first by measuring them in image and video detection datasets, and then by examining their impact on the performance of object detectors. We show that by progressively canceling out these factors, we close the gap in the performance between training on the test domain and training in the other domain. Figure 1.10 shows an example of the appearance diversity factor between videos and images. This work is published in [Kalogeiton et al., 2016]; we present it in Chapter 3, report additional results in Appendices A-B and we explore the performance of object detectors when trained from both still images and video frames in Appendix C.
- We propose to jointly detect object-action instances in uncontrolled videos, e.g., *cat eating*, see Figure 1.3. We build an end-to-end two stream network architecture (Figure 1.11), which outperforms training alone either with objects or with actions. Our architecture is able to generalize better, less prone to overfit, and benefits from sharing statistical strength between classes. We cast the joint training as a multitask objective. This boosts the joint detection perfor-

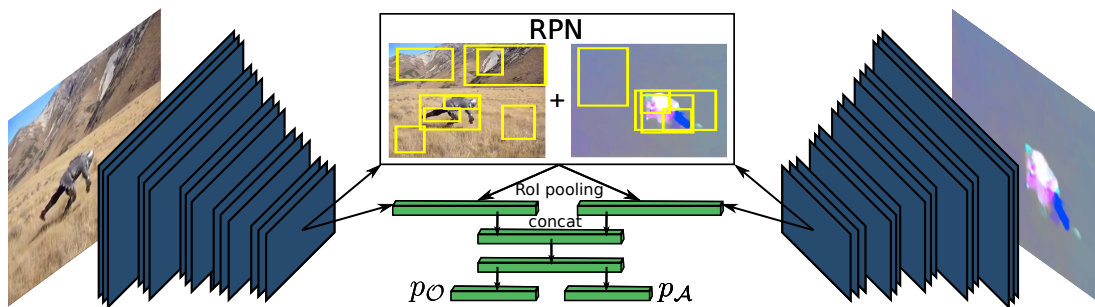


Figure 1.11: Chapter 4: Our two-stream end-to-end network for joint detection of objects and actions.

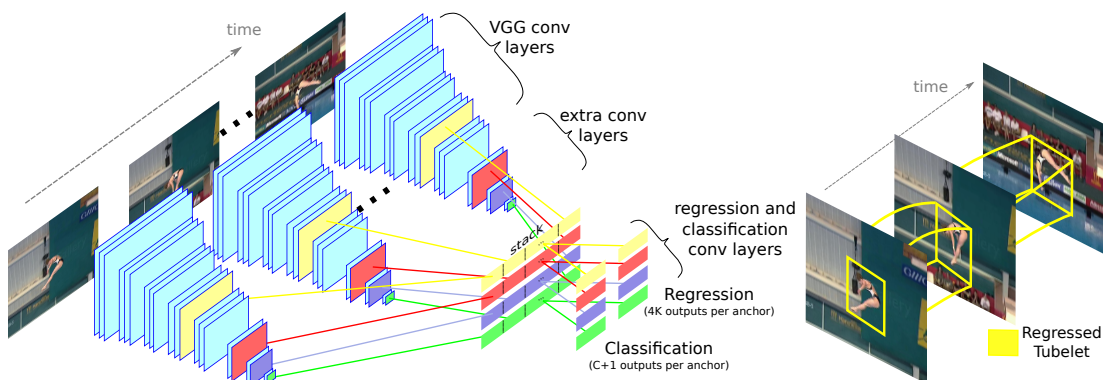


Figure 1.12: Chapter 5: Our ACT-detector takes as input sequences of frames and predicts tubelets.

mance compared to baselines and performs on par with other alternatives, such as Cartesian and hierarchical combination of objects and actions. Additionally, it requires less parameters and allows zero-shot learning of actions performed by an object. We show how to leverage our network as a generalized building block for different tasks, such as visual relationship detection or semantic segmentation, where we outperform the state of the art. This work is published in ICCV 2017 [Kalogeiton et al., 2017b]; we present it in Chapter 4, and report additional results in Appendix D.

- We tackle the action localization problem. We propose the ACTION Tubelet detector (ACT-detector), which takes as input a sequence of frames and outputs *tubelets*, i.e. sequences of bounding boxes with associated scores. Our ACT-detector is based on anchor cuboids, that handle the actor’s displacements, see Figure 1.12. We exploit the temporal information of videos by stacking features from consecutive frames and then by performing regression and classification

jointly for the whole sequence. We show that using sequences of frames instead of per-frame detections refines the spatial localization of the detections and improves their scoring, thus outperforming the state-of-the-art methods on action localization. This work is published in ICCV 2017 [Kalogeiton et al., 2017a]; we present it in Chapter 5, and report additional results in Appendix E.

**Outline.** The rest of the thesis is organized as follows: Chapter 2 presents the related work. Chapter 3 presents our exploration of domain shift factors between images and videos for object detection, Appendix A presents the second version of the YouTube-Objects dataset, and Appendix B presents additional experimental results. In Appendix C we explore the performance of object detectors when trained from both still images and video frames. Chapter 4 presents our method for joint object and action detection, and Appendix D shows additional results. Chapter 5 presents our novel action tubelet detector used for action localization, and Appendix E shows additional experimental results. Finally, Chapter 6 draws conclusions and presents perspectives for future work.





# Chapter 2

## Related work

### Contents

---

|            |   |           |
|------------|---|-----------|
| <b>2.1</b> | <b>Deep learning in computer vision . . . . .</b>         | <b>15</b> |
| 2.1.1      | Convolutional Neural Networks . . . . .                   | 15        |
| 2.1.2      | Common CNN architectures . . . . .                        | 18        |
| 2.1.3      | AlexNet . . . . .   | 20        |
| 2.1.4      | VGGNet . . . . .  | 21        |
| <b>2.2</b> | <b>Object detectors . . . . .</b>                         | <b>22</b> |
| 2.2.1      | Deformable Part-based Model . . . . .                     | 24        |
| 2.2.2      | Region-based Convolutional Neural Network . . . . .       | 26        |
| 2.2.3      | Fast R-CNN . . . . .                                      | 28        |
| 2.2.4      | Faster R-CNN . . . . .                                    | 30        |
| 2.2.5      | Single Shot MultiBox Detector . . . . .                   | 32        |
| <b>2.3</b> | <b>Domain adaptation . . . . .</b>                        | <b>34</b> |
| 2.3.1      | Adapting across different domains . . . . .               | 35        |
| 2.3.2      | Deep domain adaption . . . . .                            | 38        |
| 2.3.3      | Common architectures for deep domain adaption . . . . .   | 40        |
| 2.3.4      | Heterogeneous domain adaptation . . . . .                 | 43        |
| <b>2.4</b> | <b>Action localization . . . . .</b>                      | <b>44</b> |
| 2.4.1      | Temporal action localization . . . . .                    | 45        |
| 2.4.2      | Spatio-temporal action localization . . . . .             | 46        |
| 2.4.3      | Spatial localization for video object detection . . . . . | 50        |

**T**raditionally, a computer vision algorithm consists of two independent pipelines. The first one consists in extracting features and the second one in building machine learning techniques on top of these features to solve a given task. The first step, i.e., developing discriminative feature descriptors, has been an active area of research in computer vision for the past decades. Feature descriptors characterize the content of a given image or frame, as they aim at providing an invariant representation of videos or images with respect to their shifts, scales, luminosities, contrast etc. Some examples of hand-crafted descriptors widely used in the community are SIFT [Lowe, 1999], HOG [Dalal and Triggs, 2005], SURF [Bay et al., 2008], and Fisher Vectors [Sánchez et al., 2013].

The second step involves machine learning methods that are employed to answer the question of how to use these features as a discriminative, robust, and global video or image representation. Most methods encode these features in a robust way, for example using soft-assignment encoding [Van Gemert et al., 2010], Fisher vector encoding (FV [Sánchez et al., 2013; Perronnin et al., 2010]), or Bag Of Words (BOW) [Csurka et al., 2004], etc., such that the representation is suitable for a wide variety of tasks and for video or image types. In the past decades, these methods have led to impressive results on various tasks, such as image or video classification [Perronnin et al., 2010], video surveillance [Hu et al., 2004; Oh et al., 2011b], video captioning [Venugopalan et al., 2015; Yao et al., 2015], action recognition [Blank et al., 2005; Schuldt et al., 2004], large scale image retrieval, and classification [Philbin et al., 2007; Perronnin et al., 2010].

This two-step pipeline, however, has two major disadvantages. The first one lies in the feature extraction: the descriptors are hand-crafted. Their design is based on exploiting prior domain knowledge to make them invariant to certain properties, such as rotation, illumination, and scale, while preserving information about other properties. The second disadvantage is more subtle, as it lies in the combination of these steps: the two steps are optimized independently. This entails that the overall model – architecture and parameters – might be suboptimal.

An alternative direction is to learn features from data *for* a given task. This direction was supported by the neural network community [LeCun et al., 1989], but until recently [Krizhevsky et al., 2012] it was not used due to its huge need for training data and to limited computing resources. In particular, thanks to its impressive re-

sults, [Krizhevsky et al. \[2012\]](#) mark the revolution point of using end-to-end trainable Convolutional Neural Networks (CNNs) in computer vision.

**Outline.** In this thesis, we mainly work on CNN-based systems. Therefore, we start this chapter by a brief overview of the recent advances of deep learning, and in particular, of CNNs for computer vision (Section 2.1). Then, we divide the related work by the task they address: first we introduce the major object detectors (Section 2.2), then we review modern works on domain adaptation (Section 2.3), and finally we discuss action localization (Section 2.4). Note that in most cases we focus on the advances of the field after the arrival of CNNs.

## 2.1 Deep learning in computer vision

A CNN [[LeCun et al., 1998, 1989](#)] is a sequence of layers of neurons stacked together. Each layer transforms one volume of activations to another one through a differentiable function. We use mainly three types of layers: (a) convolutional layers (*conv* layers), (b) fully-connected layers (*fc* layers), and (c) pooling layers. The layers transform the input image volume to an output volume through a series of hidden layers. Each hidden layer is made up of a set of neurons, and applies a linear transformation to its input, i.e., convolution for *conv* layers and dot-product for *fc* layers. This linear transformation is optionally followed by a non-linearity, e.g., ReLU, sigmoid. The input and output of each hidden layer are sets of arrays called feature maps. Each layer may have some parameters and some additional hyperparameters. In comparison to a standard neural network, the CNN assumes the input to have spatial structures, such as correlations in nearby pixels of an image. We refer to the number of filters of a layer as the width of network, and to the number of layers as its depth.

**Outline.** In this section, we first describe some basic layers for CNNs (Section 2.1.1), then we introduce the most common CNN architectures used in computer vision (Section 2.1.2), and finally we describe the two architectures we use in the rest of the thesis: AlexNet (Section 2.1.3) and VGGNet (Section 2.1.4).

### 2.1.1 Convolutional Neural Networks

Here, we describe the building blocks of a CNN architecture: the *conv*, *fc* and pooling layers as well as the Rectified Linear Unit (ReLU) activation function.

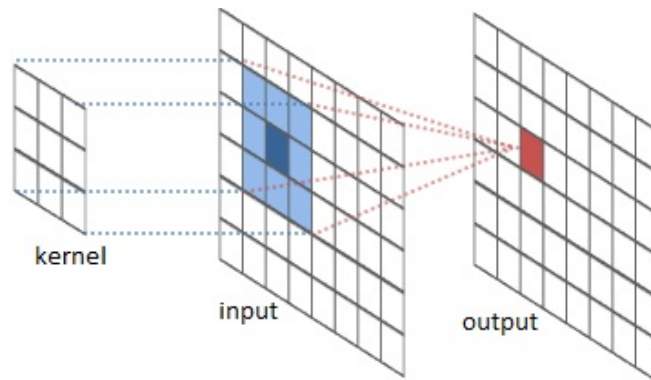


Figure 2.1: Example of convolution operation. Given an image (input) and a filter (kernel), the convolution operation computes the weighted sum of a pixel and its neighbors (dark blue and blue) and produces the output pixel value (red in output). Image source <sup>1</sup>

**Convolutional layer.** *Conv* layers are the core building blocks of a CNN, as they do the computationally heavy work (Figure 2.1). A *conv* layer consists of a set of learnable 3D filters. Typically each filter has a small spatial extent (width and height), but it extends through the full depth of the input feature map. For instance, the first filter of a *conv* layer might have dimension  $5 \times 5 \times 3$ , given that the input image has three channels (RGB). During the forward pass, we slide (or convolve) each filter across the width and height of the input feature map, and compute the dot product with the entries of the feature map. This produces a separate two-dimensional feature map for each filter. Each activation map gives the responses of its filter at every spatial location. Intuitively, the filters learn to respond to some type of visual feature such as an edge, a corner, a face, etc. Then, by stacking these activation maps along the depth dimension, we produce the output feature map to which we add a bias. This sliding window nature of the convolution operation offers to the network invariance to translation.

We need to set some hyperparameters for each *conv* layer. First, we set the filter size  $F$  (also called receptive field), which shows the spatial extent of the filter. We also set  $K$  to be the number of the filters, that corresponds to the depth of the output feature map. Then, we set the stride value  $S$  with which we slide each filter over the input feature map. When  $S > 1$ , we produce a spatially downsampled feature map, as the filters jump  $S$  pixels at a time. Finally, we set the number  $P$  that corresponds to the number of zero pixels with which to pad the input feature map. With padding we control the size of the output feature map, and usually we use it to preserve the spatial

<sup>1</sup><http://colah.github.io/posts/2014-07-Understanding-Convolutions/>

size of the input feature map, so the input and output width and height are the same.

The size of the output feature map is  $M \times M \times K$ , with  $M = (W - F + 2P)/S + 1$ , and  $W$  being the spatial size of the input feature map ( $W \times W \times D$ ). For example, in the AlexNet architecture of [Krizhevsky et al., 2012], the input image is downsampled to a size of  $227 \times 227 \times 3$ . The first *conv* layer (*conv1*) of the network has receptive field  $F = 11$ , stride  $S = 4$ , padding  $P = 0$ , and depth  $K = 96$ . The output of *conv1* is  $55 \times 55 \times 96$ .

**$1 \times 1$  convolution.** Some modern CNN architectures use  $1 \times 1$  convolutions, i.e.,  $F = 1$  as first proposed by Lin et al. [2014a]. This indicates that the filters do not take into account the spatial information in the input feature map, and hence we do not learn any visual pattern. Instead, the filters pool the information across feature maps in depth. For example, performing  $1 \times 1$  convolution with  $F = 20$  on an input feature map of  $227 \times 227 \times 300$  would result in size of  $227 \times 227 \times 20$ . Thus, the benefit of  $1 \times 1$  convolutions is that they can perform dimensionality reduction, as the depth output equals to the number of filters.

**Fully connected layer.** Fully connected layers (*fc*) connect all activations in the input feature map to each activation in the output feature map. These activations are typically computed with a matrix multiplication.

Modern CNN architectures use *fc* layers for classification. The output of the convolutional (and pooling layers) represent high-level features. Most of these features may be good for classification, but their combination might be even better. Therefore, typically the last layer of a CNN is a *fc* layer, as it is (usually) a cheap way of learning non-linear combinations of its input features. The last *fc* layer for classifiers uses softmax as the activation function, as it converts an input vector to probabilities summing to one.

Note that the only difference between *fc* and *conv* layers is that the neurons in the *conv* layer are connected only to a local region of the input, and many of them share parameters. Besides that, both layers compute dot products, and hence, any *fc* layer can be rewritten as a *conv* layer and vice versa.

**Pooling layer.** Pooling layers offer to a network robustness to small local shifts and deformations. They operate independently on each channel of an input feature map. Given an input feature map of spatial size  $W$  ( $W \times W \times D$ ), the spatial size of the output

feature map after a pooling layer is  $M \times M \times K$ , with  $M = (WF)/S + 1$ , and  $F$  and  $S$  are the filter size and the stride as in the *conv* layer, see paragraph [Convolutional layer] above. Typically, we insert a pooling layer between consecutive *conv* layers. Using pooling layers can result in downsampling the input feature map. In particular, a pooling layer with  $S \geq 2$  reduce the spatial size of the input feature map, resulting in fewer parameters and computational load. The most common pooling layer is the max-pooling. For example, a max-pooling with filters of size  $2 \times 2$  and stride  $S = 2$  takes the max over 4 numbers, therefore downsampling the input by a factor of 2 along both width and height, for every channel. Another pooling operation is the average, but it is not used so much by modern architectures, e.g. ResNet [He et al., 2016].

**Rectified Linear Unit.** Rectified Linear Unit (ReLU) is not a layer per se but an activation function. Here, we describe it as a layer, because this is how most of the community considers it.

An activation function (or non-linearity) is a fixed function that takes a single number as input and performs a fixed mathematical operation on it. It is typically applied to the outputs of a hidden layer (convolution or fully connected). There are numerous activation functions, such as sigmoid, hyperbolic tangent (tanh), etc. In computer vision, the most commonly used one is ReLU.

ReLU applies the element-wise activation function  $\max(0, x)$ , thresholding the activations  $x$  at zero. It leaves the size of the feature maps unchanged and it does not require any parameters or hyperparameters. ReLU accelerates the convergence of stochastic gradient descent compared to the sigmoid or tanh functions, e.g., by a factor of six in [Krizhevsky et al., 2012].

### 2.1.2 Common CNN architectures

There are many CNN architectures used in computer vision. The most common are:

1. LeNet by LeCun et al. [1998, 1989] is the first successful application of CNNs, aiming at recognizing digits
2. AlexNet by Krizhevsky et al. [2012] is the architecture that won the ILSVRC challenge in 2012 [Russakovsky et al., 2013a] and caused the shift of the community towards CNNs. Its architecture is very similar to the one of LeNet, but it is deeper, as it uses stacked convolutional layers on top of each other.

3. ZFNet by Zeiler and Fergus [2014] won the ILSVRC 2013 challenge [Russakovsky et al., 2013b]. ZFNet improves over AlexNet by more appropriate hyperparameters, and by having bigger size of the middle convolutional layers.
4. GoogLeNet by Szegedy et al. [2014] won the ILSVRC 2014 challenge [Russakovsky et al., 2014]. Inspired by [Lin et al., 2014a], they replace the standard convolution filter with an inception module that collects the output responses of different filter sizes. Moreover, they use  $1 \times 1$  convolutions (paragraph [1  $\times$  1 convolution] in Section 2.1) for compressing the number of feature maps resulting in fewer parameters ( $4M$ ). This enables them to make their network wider without paying a significant penalty. New versions of GoogLeNet exist, such as Inception-v4 [Szegedy et al., 2016].
5. VGGNet by Simonyan and Zisserman [2015] is the runner-up in the ILSVRC 2014 challenge [Russakovsky et al., 2014]. The main benefit of VGGNet is its depth, showing that generally deeper networks perform better.
6. ResNet by He et al. [2016] won the ILSVRC 2015 challenge [Russakovsky et al., 2015b]. It heavily uses batch normalization and it lacks of fully-connected layers at the end of the network. ResNet also introduces skip connections between layers. In general, making a network deep by adding extra layers does not guarantee that the added layers will learn extra information. The residual element in ResNet instead encourages a given layer to learn something new from what the input has already encoded.

In our work, we use AlexNet and VGGNet. Some modern detectors (see Section 2.2) do not have code available for all architectures. For instance, the R-FCN [Li et al., 2016b] and SSD [Liu et al., 2016a] detectors do not have code for GoogLeNet [Szegedy et al., 2014]. Among all architectures, AlexNet is the fastest one with improved results over hand-crafted features [Benchmarks, 2016]. We use AlexNet when we are interested in relative improvements instead of the overall performance, see Chapter 3. VGGNet offers a trade-off between speed and accuracy. It performs similarly to the first versions of GoogLeNet and ResNet [Benchmarks, 2016] while requiring more training time but less memory [Huang et al., 2016a]. Also, many modern methods use VGGNet; for a fair comparison we need to use the same underlying architecture. Below we give more details for these two architectures.



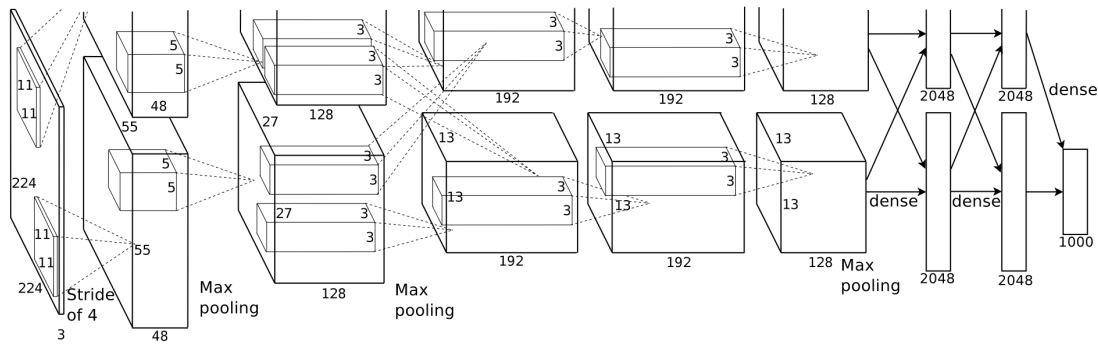


Figure 2.2: An illustration of the AlexNet architecture. It shows the split into two parts, each one operating on a different GPU. Image source [Krizhevsky et al., 2012].

### 2.1.3 AlexNet

Figure 2.2 illustrates AlexNet. It consists of eight layers, the first five are convolutional and the last three are fully-connected, see Figure 2.2. The filter sizes of the *conv* layers are, from *conv1* to *conv5*,  $11 \times 11$ ,  $5 \times 5$ ,  $3 \times 3$ ,  $3 \times 3$ ,  $3 \times 3$ . The network was initially used for image classification with 1,000 classes. In between the main eight layers there are also pooling and ReLU layers. Krizhevsky et al. [2012] found that using ReLU as the non-linearity decreases the training time compared to sigmoid and tanh non-linearities. Indeed, AlexNet has led to the widespread use of ReLU as an activation function for training deep CNNs. To address the problem of overfitting, Krizhevsky et al. [2012] also implement dropout layers [Srivastava et al., 2014].

Regarding training, Krizhevsky et al. [2012] use batch Stochastic Gradient Descent (SGD), with specific values for momentum and weight decay. During training, they also use data augmentation techniques, such as image translations, horizontal reflections, and patch extractions.

One of the specificities of AlexNet is the division of the network into two parts, see Figure 2.2. The network could not fit on a single GPU, so splitting it into two parts, with each part being executed on a different GPU, overcame the memory barrier. Additionally, the filters of the second, fourth and fifth convolution layer take as an input only those preceding feature maps that reside on the same GPU, reducing the number of learnable parameters. Moreover, this division results in good performance, as the filters learned on each GPU are independent (color agnostic and color specific).

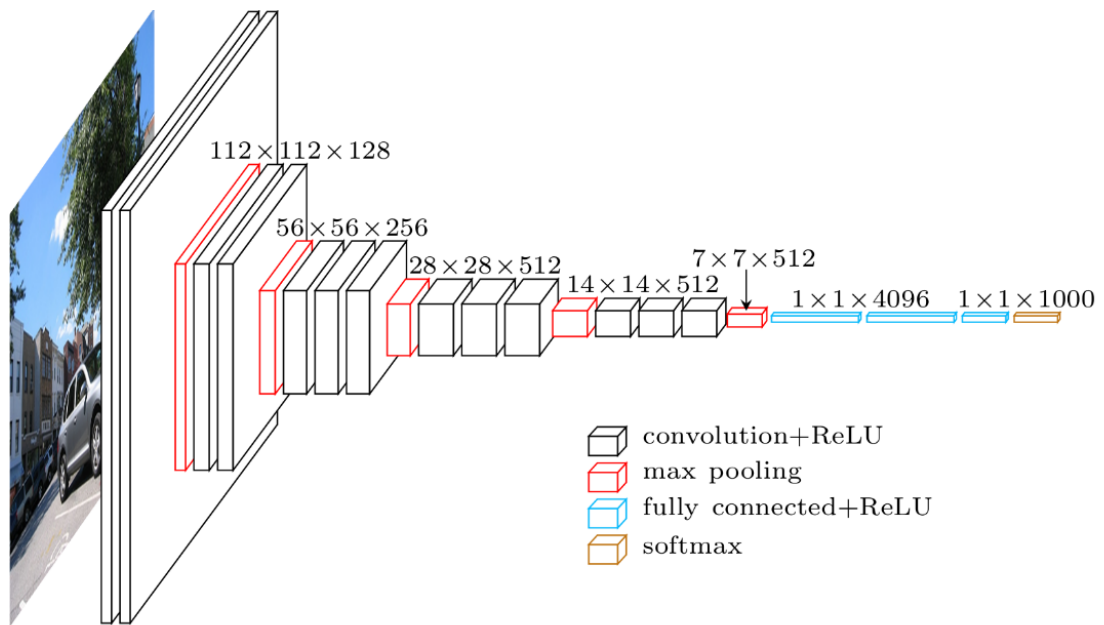


Figure 2.3: An illustration of the VGG16 architecture. Image source [Noh et al. \[2015\]](#)

### 2.1.4 VGGNet

The main characteristics of VGGNet are simplicity and depth. VGGNet is one of the most widely used architectures in computer vision as it reinforces the notion that CNNs should be deep networks to really take advantage of the hierarchical representation of visual data. [Simonyan and Zisserman \[2015\]](#) evaluate six deep CNN architectures of increasing depth up to 19 convolutional and fully-connected layers. Out of these architectures, the most commonly used one is VGG16 which has 16 convolutional and fully-connected layers.

Figure 2.3 shows VGG16. It is composed of: (a) thirteen *conv* layers (black), (b) five pooling layers (red) that perform  $2 \times 2$  max-pooling with  $S = 1$  and  $P = 1$ , and (c) three *fc* layers on top of the network (cyan). As non-linearity they use the ReLU activation function. In total the VGG16 architecture has around 138M parameters.

One of the specificities of VGGNet is that, unlike other architectures, all its *conv* layers perform  $3 \times 3$  convolutions with  $S = 1$  and  $P = 1$ . Setting the filter size  $F$  to  $3 \times 3$  removes it from the list of hyperparameters. They reason that the combination of two  $3 \times 3$  *conv* layers (without spatial pooling in between) has an effective receptive field of  $5 \times 5$ , and the combination of three  $3 \times 3$  *conv* layers has an effective receptive field of  $7 \times 7$ . Stacking three  $3 \times 3$  *conv* layers has several benefits. First, it allows using three ReLUs instead of one, making the learned output more discriminative, i.e., having a higher representational power. Second, it allows decreasing the number of

parameters, as it is a function of the filter size: given the same depth  $C$  for both the input and the output feature maps, a single  $7 \times 7$  *conv* layer requires  $7^2 C^2$  parameters, while three  $3 \times 3$  stacked *conv* layers require only  $3 \times 3^2 C^2$  parameters.

The main drawback of VGGNet is that it requires more memory compared to more recent architectures, such as ResNet, and also a high number of parameters (140M). Nevertheless, most of these parameters are in the first fully-connected layers, and it was found that removing these fully-connected layers does not reduce the performance, but it reduces the number of parameters.

## 2.2 Object detectors

Object class detection is one of the central problems in computer vision. The goal of object detection is to find the identity (class) and the spatial location of an object in an image. Therefore, one can claim that object detection is a joint classification and localization problem. Object detectors are trained from a large pool of images annotated with bounding boxes.

Over the last years, several works tackle object detection [Cinbis et al., 2014; Gonzalez-Garcia et al., 2015; Papadopoulos et al., 2016, 2017; Song et al., 2014]. Their common characteristics are that they first associate images with a set of proposals that ideally overlap with the ground-truth bounding boxes, i.e., class-generic region boxes [Alexe et al., 2010; Uijlings et al., 2013; Zitnick and Dollár, 2014], then they represent these proposals with robust feature vectors, and finally they score each proposal for each object class with a confidence. CNNs merge the feature representation and classification steps, allowing for end-to-end training.

There are various works that either build object detectors, or build upon existing object detectors, to improve detection performance. The major modern object detectors are: R-CNN by Girshick et al. [2014a], OverFeat by Sermanet et al. [2014], MultiBox by Erhan et al. [2014], SPP He et al. [2015], DeepBox by Kuo et al. [2015], MR-CNN by Gidaris and Komodakis [2015], AttentionNet by Yoo et al. [2015], DenseBox by Huang et al. [2015], Fast R-CNN by Girshick [2015a], DeepID Net by Ouyang et al. [2015], Faster R-CNN by Ren et al. [2015a], NoC by Ren et al. [2016], YOLO by Redmon et al. [2016], R-FCN by Li et al. [2016b], SSD by Liu et al. [2016a], and YOLOv2 by Redmon and Farhadi [2016].

There is a vast amount of works that use these detectors for various tasks:

1. object detection [Cinbis et al., 2014; Gonzalez-Garcia et al., 2015, 2017; Hen-

derson and Ferrari, 2016; Kalogeiton et al., 2016; Kim et al., 2014; Wang et al., 2014; Song et al., 2014; Prest et al., 2012a; Sharma and Nevatia, 2013; Tang et al., 2012; Vezhnevets and Ferrari, 2015],

2. stuff detection [Shi et al., 2017], which focuses on stuff classes, such as *grass*, *wall*, or *sky* instead of thing classes, such as *person* or *car*.
3. parts detection [Gonzalez-Garcia et al., 2016; Modolo and Ferrari, 2016], where each class is an object part, a piece of a bigger object composed of multiple parts, e.g. *arm* or *head*,
4. human pose estimation [Rogez et al., 2017], which is defined as the problem of localization of human joints,
5. weakly supervised detection [Papadopoulos et al., 2014, 2016, 2017; Shi and Ferrari, 2016], where given a set of images/videos labeled only as containing a certain object class, without being given the location of the objects, the goal is to localize the objects in these training images/videos while learning an object detector for localizing instances in new test images/videos,
6. semantic segmentation [Caesar et al., 2015, 2016; Tokmakov et al., 2017b, 2016, 2017a; Volpi and Ferrari, 2015], which aims at understanding an image/video at pixel level i.e., assign each pixel in the image/video to a meaningful object class.
7. action detection [Gkioxari and Malik, 2015; Kalogeiton et al., 2017a; Niebles et al., 2010; Peng and Schmid, 2016; Saha et al., 2016; Singh et al., 2017; Kang et al., 2016a], where the goal is to localize spatially and temporally the human actions in videos, and
8. for video alignment or discovery [Del Pero et al., 2016a,b; Papazoglou et al., 2016b,a], which usually refers to synchronization, i.e., the establishment of temporal correspondence between frames of the first and second video, followed by spatial registration of all the temporally corresponding frames.

**Outline.** In the following, we introduce the five object detectors we use in our work. First, in Section 2.2.1, we introduce the Deformable Part-based Model (DPM) of Felzenszwalb et al. [2010] that was the leading detector before the arrival of CNNs. Then, we describe all the relevant CNN-based detectors. We start with the Region-based Convolutional Neural Network (R-CNN) of Girshick et al. [2014a] (Section 2.2.2). Then,

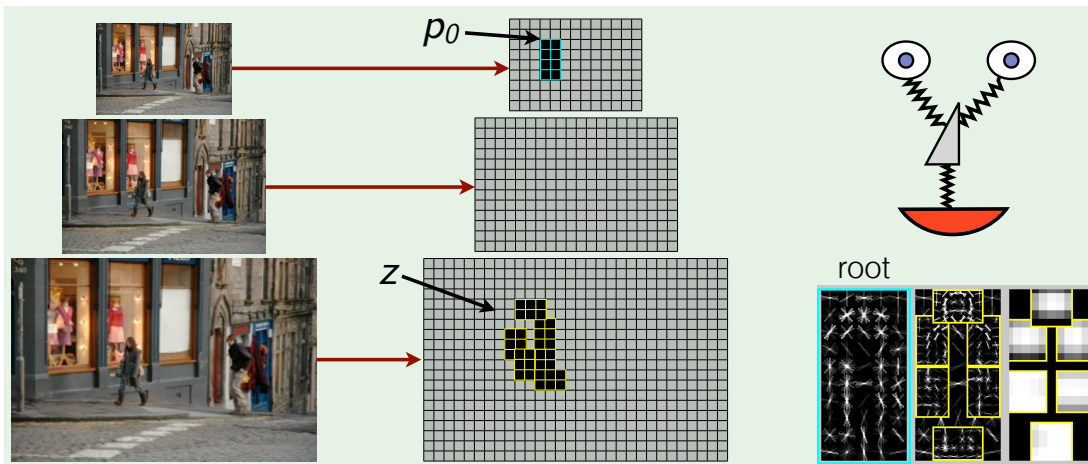


Figure 2.4: An illustration of DPM: (left) image pyramid, (middle) HOG feature pyramid, and (right) root filter, part filters and spatial models for the person class. The visualization of the spatial models reflects the cost of placing the center of a part at different locations relative to the root. Image source [Girshick et al., 2012a].

in Section 2.2.3 we describe the Fast R-CNN detector of Girshick [2015a], and its successor the Faster R-CNN detector of Ren et al. [2015a] (Section 2.2.4). Finally, in Section 2.2.5 we describe the recently proposed Single Shot MultiBox Detector (SSD) of Liu et al. [2016a].

### 2.2.1 Deformable Part-based Model

Deformable part-based model (DPM) was the leading object detector from 2009 until the publication of R-CNN [Girshick et al., 2014a]. It uses a sliding window approach based on mixtures of multi-scale deformable models. It is able to represent a high variety of classes, achieving good performance detection. The code of DPM is available online [Girshick et al., 2012b].

**Model.** Figure 2.4 illustrates the DPM detector. It is a sliding window detector, where each filter is applied at all positions and scales in the input image. Felzenszwalb et al. [2010] base their implementation on the HOG detector of Dalal and Triggs [2005]. Inspired by the pictorial framework of Felzenszwalb and Huttenlocher [2005], i.e., a collection of parts with connections between certain pairs of parts, they enrich the HOG detector with a star-structured part-based model: first with the one central part that covers the whole object, called the *root*, and second with a set of part filters and associated deformation models that are connected to the root. Intuitively, DPM repre-

sents an object by various parts arranged in a deformable configuration, that captures the relative position of each part with respect to the root.

The scores of both root and part filters are computed at different resolutions in a feature pyramid of the input image. For a given image, the part scores are also associated with a deformation cost that penalizes a part for deviating from its ideal location relative to the root. Finally, the score of a model is the sum of the root score, the maximum part score and the deformation cost.

The benefits of using parts are multifold. First, it is easier to model local than global appearance. Second, the training data can be shared across deformations. Finally, parts generalize to previously unseen configurations.

Additionally, DPM represents an object class model as a mixture of separate components, allowing to cover a wider range of appearance diversity for each object class. Each component is trained on a subset of the training data based on common characteristics, such as aspect, occlusion modes, and subclasses. The score of a mixture model is the maximum over the scores of its components.

**Training and testing.** The model is trained using a latent SVM classifier with hard-negative mining [Dalal and Triggs, 2005]. Given a negative image, the detector should not ‘fire’ in any location of this image, entailing that the score of the root filters should be small. Respectively, the resulting score of a positive image should be high. Therefore, at each iteration, DPM first adds feature vectors for each positive sample, then it collects hard negative examples, and finally removes the easy ones. Positive examples are the ones classified correctly with high confidence, and similarly negative examples are classified wrongly with high confidence. Hard negative examples are previously false positive examples that are added to the training set, and they help reducing the number of false positives. At test time, DPM produces a score for each component, based on the root scores, the parts scores and the deformation costs.

**Discussion.** DPM was the state-of-the-art detector for many years. It revived the pictorial structure of Felzenszwalb and Huttenlocher [2005] by taking advantage of modern (for its period) features (HOG [Dalal and Triggs, 2005]) and statistical methods (latent SVMs). DPM is trained with bounding boxes and outputs object detections, and part detections.

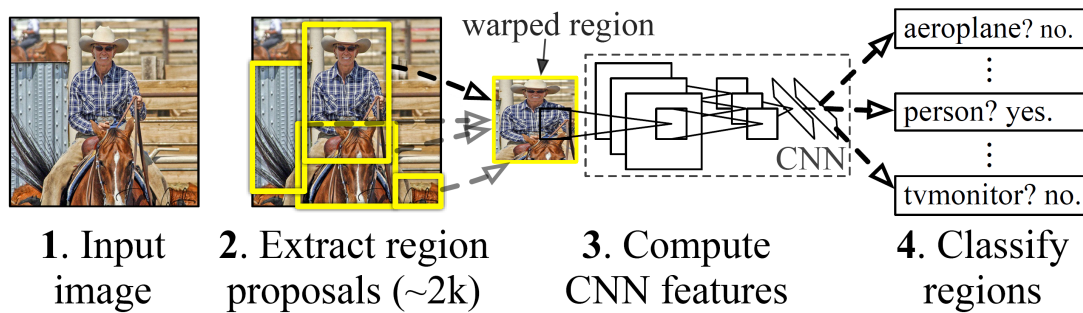


Figure 2.5: An illustration of the R-CNN detector. For a test image, (1) R-CNN extracts class-generic region proposals, (2) computes their CNN features, and (3) classifies each proposal using class-specific linear SVMs (4). Image source [Girshick et al., 2014a].

## 2.2.2 Region-based Convolutional Neural Network

Region-based Convolutional Neural Network detector (R-CNN) is the first work that brought the deep learning framework into the object detection field. It was proposed by Girshick et al. [2014a], achieving state-of-the-art results on modern image datasets [Everingham et al., 2007, 2012]. The code of R-CNN is publicly available in [Girshick et al., 2014b], and is based on the Caffe deep learning framework [Jia et al., 2014; Jia, 2013].

**Model.** Figure 2.5 illustrates the R-CNN detector, consisting of three steps. For an input image, they generate class-generic object proposals using [Uijlings et al., 2013]. Then, they represent each object proposal with a fixed-length feature vector computed by a CNN [Krizhevsky et al., 2012]. Finally, they classify each proposal. Following the ‘recognition using regions’ paradigm [Gu et al., 2009], which was based on [Alexe et al., 2010; Uijlings et al., 2013], R-CNN generates class-generic region proposals for each image that are then classified using class-specific linear SVMs. Region proposal techniques replace the sliding-window paradigm which extracts millions of regions from an image, with extracting only a few thousands class-generic regions from an image, likely to contain an object [Alexe et al., 2010; Uijlings et al., 2013]. The key novelty of R-CNN is using CNN architectures as the underlying feature extraction mechanism to represent each region proposal prior to its classification. This replaces hand-crafted features, such as HOG [Dalal and Triggs, 2005], with high-level object representations coming from a CNN architecture. As the CNN is specifically trained for region classification, it leads to to very discriminative features for each region.



**Training and testing.** R-CNN uses AlexNet as the underlying CNN architecture. At the end of AlexNet, they add a fully-connected  $fc$  softmax layer (see paragraph [Fully connected layer] in Section 2.1), that outputs a probability over the number of classes plus one for the background class, i.e., an additional class for when there is no object in the region. Training R-CNN means fine-tuning the AlexNet for region classification. This key process results in discriminative and robust features for each region proposal. Note that AlexNet is already pre-trained on the ILSVRC12 classification challenge [Krizhevsky et al., 2012], and Girshick et al. [2014a] use the 7<sup>th</sup> layer of the CNN as features. R-CNN also uses hard negative mining when training the SVM classifiers, like DPM. Here, however, the hard negative mining also comes from the region proposals. In particular, at each iteration 25% of the region proposals are positive ( $\text{IoU}^2 \geq 0.5$  with a ground-truth bounding box) and the rest are negative ( $\text{IoU} < 0.3$  with a ground-truth bounding box).

At test time, R-CNN first extracts around  $2k$  region proposals for each image that are then resized (warped) to  $227 \times 227$ . Each proposal is then propagated forward through the fine-tuned AlexNet to get CNN features. After that, for each class, Girshick et al. [2014a] score all proposals using the SVM trained for that class. Finally, given all scored regions in an image, the authors apply a greedy non-maximum suppression algorithm (NMS) for each class independently. For both training and testing, they use Selective Search [Uijlings et al., 2013] to generate object proposals.

**Discussion.** R-CNN bridged object detection with deep learning, but it has some clear disadvantages. First, it has some ad hoc objectives, such as fine-tuning the network with softmax classifiers, and training post hoc SVM classifiers and bounding-box regressors. Second, it exhibits noticeable inefficiencies in terms of both speed and storage. Training takes up to 100 hours for around  $10k$  images and requires lots of gigabytes of disk space to save the features. Speed is also an issue at test time. For instance, when using VGGNet, inference takes around  $47s$  per image, i.e., around  $65h$  for a small image dataset like Pascal VOC 2007 [Everingham et al., 2007], and more than  $100h$  for small video datasets.

Of course, there are some important principles introduced by R-CNN, that most modern works apply nowadays. The first one worth mentioning is using pre-trained

---

<sup>2</sup>Everingham et al. [2010] proposed the metric Intersection Over Union (IoU): a predicted bounding-box (the detection) is considered ‘correct’ if its intersection with a ground truth bounding-box (the real bounding-box) divided by their union is greater than a threshold (usually set to 0.5). It is also called PASCAL VOC criterion ( $\text{IoU} > 50\%$ ).



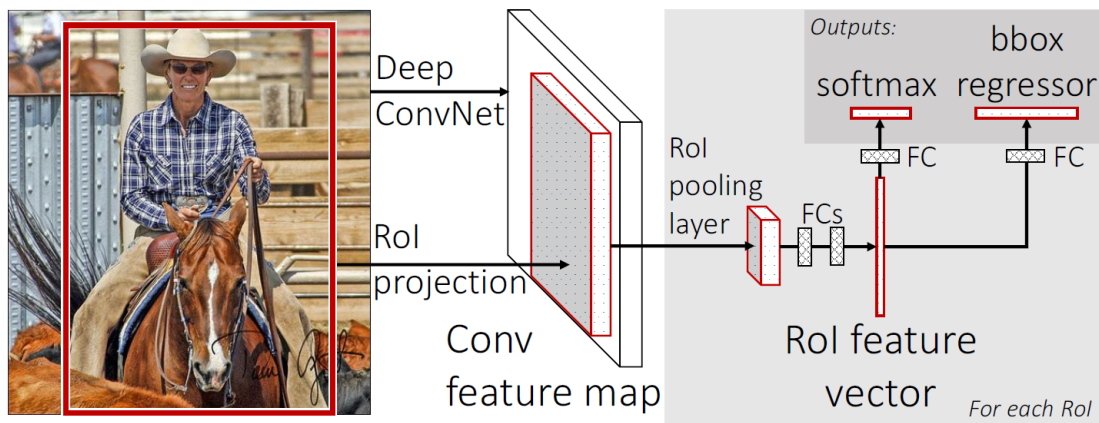


Figure 2.6: An illustration of the Fast R-CNN model: the input image and multiple RoIs are fed into a fully-convolutional network. Each RoI is pooled and mapped to a feature vector. The network branches into two layers, a softmax layer, that outputs softmax probabilities, and a bounding box regression layer, that predicts the four coordinates of the bounding boxes. Image source [Girshick, 2015a].

models. This is a key insight, as it transfers information from images of another data-rich task (classification) to detection, where the size of the datasets is either limited (in images) or extremely redundant (in videos). Overall, R-CNN represented a remarkable step forward for object detection.

### 2.2.3 Fast R-CNN

Fast R-CNN [Girshick, 2015a] was introduced in 2015 and is a fast follow-up of R-CNN. At test time, it processes images almost in real time, without considering the time to extract object proposals. It employs several innovations mainly to improve training and test speed while also increasing detection accuracy. It is a fast detector, trained in one stage without taking storage space. It therefore surpasses the speed and storage limitations of R-CNN, see paragraph [Discussion] in Section 2.2.2. The code of Fast R-CNN is available online [Girshick, 2015b].

**Model.** Figure 2.6 illustrates the Fast R-CNN model. It takes as input the entire image and all object region proposals produced by Selective Search [Uijlings et al., 2013]. The full image is propagated through the whole network up to the newly-introduced Region of Interest (RoI) pooling layer, which is placed after the last *conv* layers. The RoI layer operates in the proposal domain: given the feature map of the whole image, it extracts a fixed-length feature vector for each proposal. The RoI pooling layer uses

max-pooling to convert the *conv* feature maps into small feature maps for every valid region of interest. Each feature vector is then fed into the *fc* layers of the network. The *fc* layers map each RoI to a feature vector. In the end, the network branches into two layers, a softmax layer, the same as in R-CNN, outputs softmax probabilities and another layer, that predicts the four coordinates of the bounding boxes for all classes.

**Training and testing.** Training Fast R-CNN is equivalent to fine-tuning it (as in R-CNN), as the underlying network being used is already pre-trained. Girshick [2015a] proposes an efficient method relying on feature sharing at training time. First, they sample  $N$  input images, and then they sample  $R/N$  RoI per image (called mini-batch sampling). The key here is that the sampled RoIs from the same image share computation and memory. In their experiments, they use only  $N = 2$  images at each training step, with  $R = 128$ , i.e., 64 RoIs per image. For each mini-batch sampling, they use the same hard-negative mining as R-CNN. The only data augmentation technique used is horizontally flipping images with a probability of 0.5.

At test time, they extract object proposals for each image and they forward them into the network. The two final branches of the network output a set of regressed proposals with their associated scores.

**Discussion.** The benefits of Fast R-CNN are its speed, accuracy, and not requiring disk storage. It is fast:  $\times 9$  faster at training and  $\times 213$  faster at test time compared to R-CNN. Additionally, it achieves higher mAP than the previous detectors. Note here, that Fast R-CNN is also inspired by SPP [He et al., 2015], as the RoI pooling layer is actually a special case of SPP layer. Nevertheless, we do not review SPP here as we do not use it in our work. We should note though that the main advantage of the RoI pooling layer is offering translation invariance. Fast R-CNN has two losses (one for classification and one for regression) trained in a multi-task manner. This joint optimization is found to work better than optimizing objectives independently. Finally, Fast R-CNN is a simpler implementation compared to R-CNN and it does not require persistent storage of intermediate features. Its main drawback is that the region proposals are not part of the network and their generation also requires some time, typically around 2s per image.

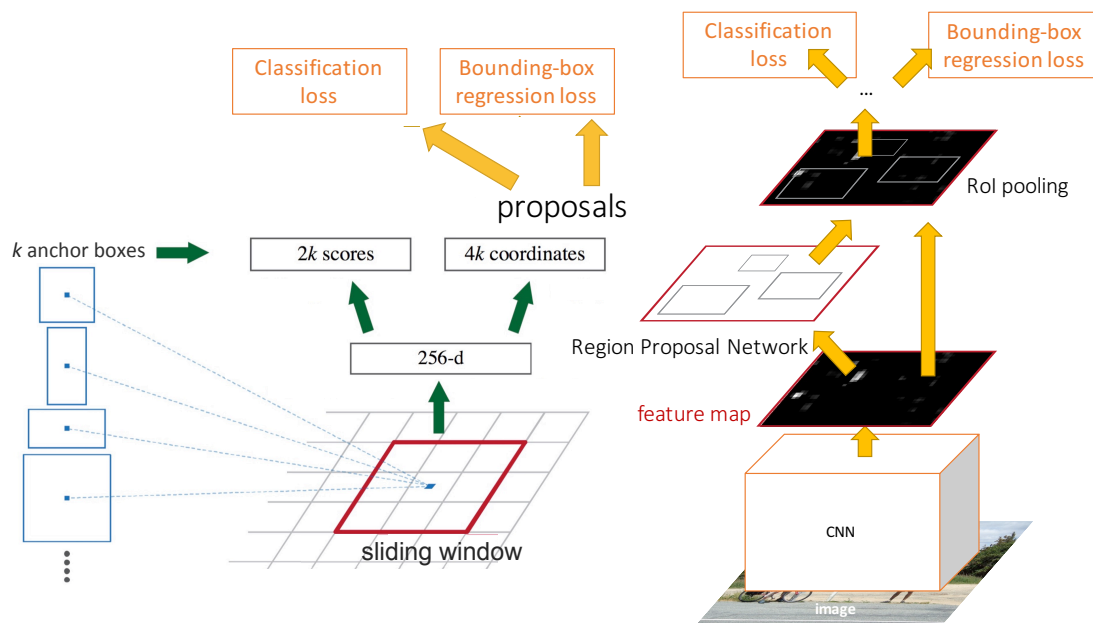


Figure 2.7: An illustration of the Faster R-CNN detector. One network with four losses. The first two losses belong to the RPN. Image source [Ren et al., 2015a].

## 2.2.4 Faster R-CNN

Faster R-CNN is the latest update of the R-CNN series. It was proposed by Ren et al. [2015a], who built upon Fast R-CNN with a major improvement: the region proposal generation becomes an independent CNN component of the detector, called Region Proposal Network (RPN). It achieved state-of-the-art results on object detection [Everingham et al., 2007, 2012]. The code of Faster R-CNN is also available online [Ren et al., 2015b].

**Model.** Figure 2.7 illustrates the Faster R-CNN detector. The model is the same as Fast R-CNN with an extra component: the RPN is used to generate proposal regions. The RPN is placed after the last shared convolutional layer, i.e., *conv5*, and slides over the feature map to determine whether a region is an object or not. Note that the RPN layers are shared with the object detection layers.

The image is fed into the network and after *conv5* the convolutional map goes to the RPN. A small, typically  $3 \times 3$ , *conv* layer slides across the input feature map. Each candidate region is mapped to a lower-dimensional feature vector that is then fed to two consecutive *fc* layers. The first one is a two-class classification layer indicating if the region is an object or not. The second one is a regression layer that adjusts the spatial position of the bounding box.

The RPN relies on *anchor* boxes. At each sliding region, the RPN simultaneously predicts multiple region proposals of fixed position and aspect ratio with two associated scores, and regressed coordinates. These proposals are called anchors. An anchor is centered at the sliding region and represents a combination of different scales and aspect ratios, so multiple anchors correspond to the same receptive field. In Faster R-CNN, there are in total three fixed scales and three fixed aspect ratios, resulting in nine anchor boxes at each sliding region.

Similarly to Fast R-CNN, each of the generated proposals is then passed to the RoI pooling layer, which makes the network more translation-invariant. The layer size varies with the size of input feature map, but the output is a fixed feature vector. The RoI output is fed to a *fc* layer, which performs classification and regression as in Fast R-CNN, see paragraph [Model] in Section 2.2.3.

**Training and testing.** As for its ancestors, training Faster R-CNN is equivalent to fine-tuning it. First, the network is initialized with the ILSVRC12 pre-trained model. Then, they fine-tune the whole network, including the RPN. After both the RPN and Fast R-CNN are trained, their *conv* layers are shared in order to train one unified network. The Fast R-CNN network is trained following the method described in paragraph [Training and testing] of Fast R-CNN, i.e., sampling RoIs from each image and using hard-negative mining.

Given, however, that most anchor boxes are negative, they only keep a subset of them, so that the overall training set consists of 25% positive and 75% negative boxes.

There are two main techniques for this training introduced by [Ren et al. \[2015a\]](#). The first one consists in alternating between training of the RPN and Fast R-CNN. The training steps are: fine-tuning the RPN, use the RPN to fine-tune Fast R-CNN, use Fast R-CNN to re-initialize the RPN, and finally fine-tuning the layers unique to then RPN by keeping fixed the shared *conv* layers. The second technique is the approximate joint training, where RPN and Fast R-CNN are merged into one network during training. The proposals generated by the RPN are treated as fixed by Fast R-CNN. The first technique yields higher results, but the second one reduces the training time by up to half, while achieving similar results. For this reason, in our work we use the approximate joint training, see Section 4.

Finally, at test time, they feed the input images into the unified network. The two final branches of the network output a set of regressed proposals with their associated scores.

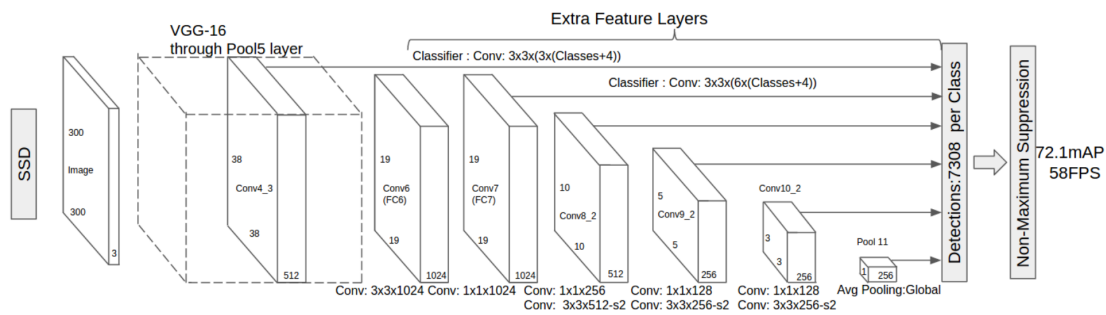


Figure 2.8: An illustration of SSD. It adds several conv layers to the end of a base network. The extra conv layers predict offsets to anchor boxes of different scales and aspect ratios and their associated confidences. Image source [Liu et al., 2016a].

**Discussion.** Faster R-CNN sheds the region proposal limitations inherited by R-CNN and Fast R-CNN with the introduction of the RPN. Faster R-CNN is faster than Fast R-CNN when summing up the time for externally generating proposals and training the network. An interesting part in Faster R-CNN is the anchor boxes, which is used by upcoming modern detectors.

## 2.2.5 Single Shot MultiBox Detector

In 2016, Liu et al. [2016a] proposed the Single Shot MultiBox Detector (SSD). It uses a fully convolutional architecture (no fully-connected layers) that relies on anchor boxes of different scales and aspect ratios, eliminating the need for any object proposal generator, and thus enabling fast computation. It is currently the state-of-the-art detector and its code is publicly available [Liu et al., 2016b].

**Model.** Figure 2.8 illustrates SSD. It discretizes the output space of bounding boxes into a set of anchor boxes (called *default* boxes) over different aspect scales and aspect ratios per feature map location. SSD has two core components. The first one, the base network, is any standard architecture, up to its last *conv* layer (typically *conv5*, see Sections 2.2.3-2.2.4). This base network is used for high quality image classification. The second component is a list of stacked *conv* layers on top of the base network that progressively decrease in size. The key characteristic is that each added *conv* layer produces a fixed set of detection predictions using convolutional filters. The different resolution of these layers allows predictions at multiple scales.

The feature map of each *conv* layer is associated with a set of anchor bounding boxes. In particular, for each anchor box at any given location one classification score

for each class and four regressed coordinates are computed. SSD's anchor boxes differ from the ones of Faster R-CNN because the boxes of the former are applied to several feature maps of different resolutions. This makes the receptive fields of the anchor boxes of SSD more relevant than the ones of Faster R-CNN.

**Training and testing.** At training, they use various data augmentation techniques, making the detector more robust to various input object sizes and shapes. Given the input image, they randomly sample patches from it, which are then either resized, flipped, transformed with photo-metric distortions, or kept as they are. They also introduce another data augmentation technique: a *zoom out* operation, where they first randomly place the input image on a canvas of sixteen times the original image size, filled with the mean value of the input image, and then they randomly crop parts of the image. They show that these data augmentation techniques bring a noticeable increment (around 2%) in the detection performance. For this reason, in Section 5, we also adopt it.

The required input of SSD is the image with the ground-truth bounding boxes. SSD matches each ground-truth box first with the anchor box that best overlaps with it, and then with all the other anchor boxes that have  $IoU \geq 0.5$ . This matching defines the positive training set, and it allows the network to predict high scores for multiple overlapping anchor boxes, instead of relying only on the one with maximum overlap. Given, however, that most anchor boxes are negative, they only keep a subset of them, so that the overall training set consists of 25% positive and 75% negative boxes.

At test time, they also apply the same data augmentation techniques and predict regressed bounding boxes with their associated scores.

**Discussion.** SSD brings three contributions. The first one is that the anchor boxes are more relevant than the ones of Faster R-CNN, as they are of different scales and aspect ratios and are coming from layers of different scales. The second one is its speed: as it is fully convolutional without any proposal generator, it achieves state-of-the-art results while requiring less computational time than other detectors. The last and most important one is allowing predictions from different output layers of different resolution.

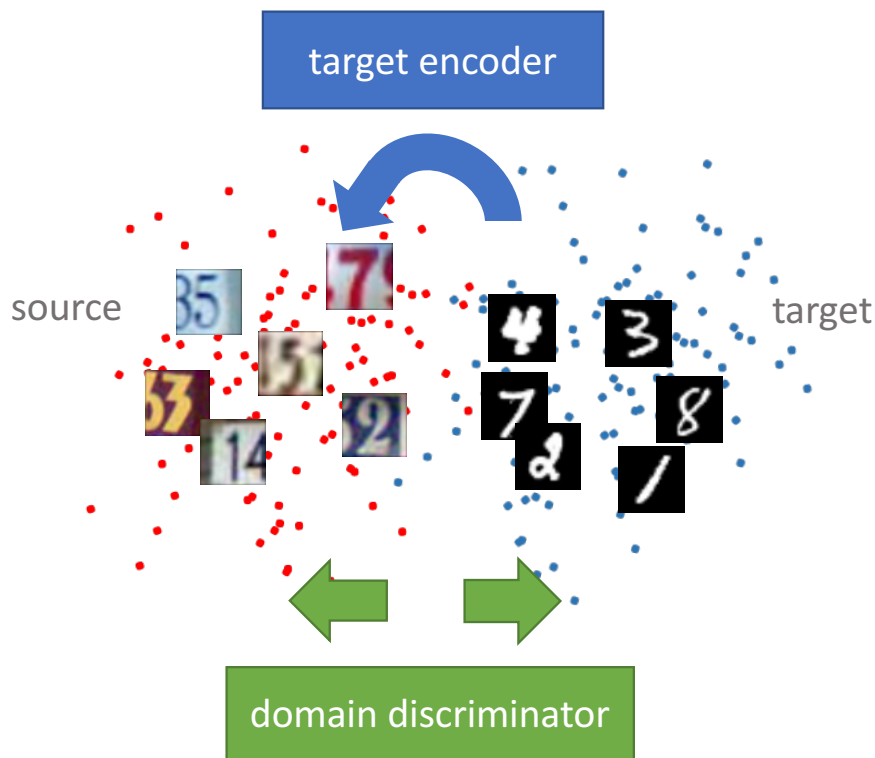


Figure 2.9: Example of domain adaptation, where goal is to learn a discriminative mapping of target images to the source feature space (target encoder) by fooling a domain discriminator, which tries to distinguish the encoded target images from source examples. Image source [Tzeng et al., 2017].

## 2.3 Domain adaptation

Supervised learning consists in making predictions for some test data using information from training data. It assumes that both the training and test data are drawn from the same distribution, which, for many scenarios, is unreasonable to assume. Domain adaptation enables the training and testing distributions to be different (also called distribution shift) [Jiang, 2008; Pan and Yang, 2010]. The goal of domain adaptation is to learn to adapt the source domain to the target one so as to improve the target performance. For instance, the knowledge obtained when learning to recognize apples might help when learning to recognize oranges. Figure 2.9 illustrates the common goal of modern domain adaptation techniques.

Typically, the source domain represents the training data, and the target domain represents the test data. The training data are labeled, while the test data are either partially labeled (supervised domain adaptation) or unlabeled (unsupervised domain

adaptation). Supervised domain adaptation assumes having access to only few labels for the test data, not enough though to train a robust model. Therefore, supervised domain adaptation methods learn either how to adapt the source domain to the target, or how to regularize the target domain given the source data. Unsupervised domain adaptation assumes that the test data have no labels, hence focusing on aligning the train and test distributions [Aljundi and Tuytelaars, 2016] through sample re-weighting [Fernando et al., 2013; Raj et al., 2015] or non-linear transformations [Sun and Saenko, 2016].

The most common case of domain adaptation in computer vision is adapting across datasets. This can mean learning information from one dataset and generalizing to others, e.g., pre-training on ILSVRC [Russakovsky et al., 2013a] and fine-tuning on VOC [Everingham et al., 2007]. It can also mean that the modality of the datasets is different, e.g., adapting from images to videos. Before the arrival of CNNs, most domain adaptation approaches were divided into two categories. The works of the first category aim to learn an invariant transformation of the source features [Daumé III, 2009; Quadrianto and Lampert, 2011; Sharma et al., 2012a], while the second ones aim to find a representation where the distance between the source and the target distributions is minimized [Bergamo and Torresani, 2010; Duan et al., 2009, 2012c; Yang et al., 2007b]. Modern methods rely on CNNs and therefore improve both steps simultaneously: learning how to robustly project the features of both domains in a common feature space, while at the same time learning the parameters of their distributions [Ganin and Lempitsky, 2015; Long et al., 2015].

**Outline.** In Section 2.3.1, we first review the related work on adaptation before the arrival of CNNs. In Section 2.3.2 we discuss the deep domain adaptation, and then in Section 2.3.3 we discuss common domain adaptation methods, such as discrepancy-based or adversarially-based methods. Finally, in Section 2.3.4 we review the related work when the modality of the source and target domains is different, such as RGB images, depth images, videos, etc.

### 2.3.1 Adapting across different domains

There are plenty of factors (alone or combined) that make the distributions of two domains differ. Differences can be due to biases in the distribution of viewpoints, scenes, intra-category variations, poses, illumination conditions, cluttered background, blur–



especially motion blur-, articulation, camera framing, resolution, spatial location of the object, occlusion patterns, camera characteristics such as lens, etc. Studies [Torralba and Efros, 2011] have shown that datasets are biased due to the different way they are collected and annotated.

Domain adaptation learns the shift in the distributions between training (source) and test (target) data. We divide the domain adaptation approaches into three categories: learning adapted classifier parameters, feature transformation across domains, and combinations of the two.

**Approaches based on adapted classifiers.** There are many approaches that focus on learning how to adapt classifiers (typically SVMs) to the target data given the source data. These methods rely on the existing feature representation of the data. For instance, Bergamo and Torresani [2010] propose a specific domain adaptation transductive SVM algorithm to learn examples in the target domain by generalizing the source domain. Adaptive SVMs are proposed by Yang et al. [2007a,b], who adapt an existing auxiliary classifier to a target classifier by learning a perturbation function between the decision functions of the two classifiers. They learn the target classifier using an extension of the standard SVM's objective function, looking for a decision boundary that achieves a small classification error and creates a large margin between training examples of any two classes. Aytar and Zisserman [2011] extend the adaptive SVM to a model called projective model transfer SVM using a different regularization term that does not indirectly penalize the margin. In this way, they regularize the training of new categories, given a model trained on some categories.

Duan et al. [2009] propose a domain transfer SVM, a cross-domain kernel method, that learns both a kernel function and an SVM classifier. They measure the mismatch between the source and target distributions using their maximum mean discrepancy, and they minimize it while learning the target decision function. The same group [Duan et al., 2012c] later suggested learning a kernel function based on multiple base kernels using multiple adaptive kernel functions.

Most of these methods are unable to transfer the learned domain shift to unseen classes, which may be required for new datasets with unknown classes.

**Approaches based on feature transformation.** Most feature adaptation methods use existing classifiers, but learn a transformation between the features of the source and target domains. For example, Daumé III [2009] essentially forces the learning

algorithm to do the adaptation by augmenting the feature space of the source data using some target samples.

To learn the feature transformation between domains, there are some methods that exploit corresponding points between the source and target domains. For instance, the method of [Saenko et al. \[2010\]](#) learns a regularized non-linear transformation that maps feature points from the source domain to the target domain using cross-domain constraints. The constraints enforce that the transformation maps points from the same class (but different domain) close to each other. The output can be applied to previously unseen target data, or even classes not seen at training time. The same group later presented Asymmetric Regularized Cross-domain Transform (ARC-T) [[Kulis et al., 2011](#)] to encode the domain invariance into the feature representation. ARC-T learns from the labels of a class, and adapts the models between heterogeneous spaces of different dimensionality via an asymmetric transformation.

Also, several works exploit corresponding points coming from multi-view data. For example, the multi-view learning approaches [[Quadrianto and Lampert, 2011](#); [Sharma et al., 2012a](#)] use multiple sets of observations for the same instance to learn the target feature representation. Moreover, [Sharma et al. \[2012a\]](#) propose Generalized Multi-view Analysis (GMA), a general framework to learn a common discriminative subspace by simultaneously learning multi-view projection directions and generalizing their model across unseen classes. Other multi-view approaches were proposed by [Farhadi and Tabrizi \[2008\]](#) and [Li and Zickler \[2012\]](#), who translate features from activity scenes captured with one camera view to learn discriminative models and use them to spot the same activity in another view. Finally, [Kan et al. \[2016\]](#) propose Multi-view Discriminant Analysis (MvDA), which projects the features from different views to a single discriminant common feature space by jointly maximizing the inter-class variations and minimizing the intra-class variations.

Some other methods learn the feature transformation by aligning the domain distributions in an unsupervised setting [[Gong et al., 2012](#); [Gopalan et al., 2011](#); [Mittal et al., 2016](#)]. For example, [Gopalan et al. \[2011\]](#) considers that starting from the source domain we can reach the target domain by following a manifold path of finite sets of subspaces. They build these subspaces and consider as feature representation of each one the concatenation of intermediate subspaces. Similarly, the method of [Gong et al. \[2012\]](#) considers that the path between the source and the target domains consists of subspaces. They propose a geodesic flow kernel and model this path with an infinite number of subspaces that characterize shifts in geometric and statistical properties.

Additionally, given multiple source domains and only one target domain, they are able to select the optimal source domain on the geodesic path in order to reach the target domain.

**Approaches for jointly adapting classifiers and features.** There are some methods [Duan et al., 2012b; Hoffman et al., 2013] that aim at both adapting the classifiers and learning a feature transformation.

The Heterogeneous Feature Augmentation method HFA [Duan et al., 2012b] first measures the difference in the feature distributions of data coming from two heterogeneous domains and then maps them to a common subspace. Then, Duan et al. [2012b] introduce an objective function for a classifier to learn from the two data representations. HFA learns a separate feature representation for each class, and therefore it does not generalize to new classes. Similarly, the Maximum Margin Domain Transform method (MMDT) [Hoffman et al., 2013] jointly adapts max-margin classifiers in a multi-class manner and learns the feature transformation to capture the domain shift. MMDT allows efficient optimization in linear space by learning a transformation directly from the source to the target domain. The feature representation learned by MMDT generalizes to novel target categories without additional computational cost.

### 2.3.2 Deep domain adaption

Deep learning methods allow to learn simultaneously features and model parameters for a given task. Supervised learning with CNNs requires a large number of labeled data for training. Chopra et al. [2013] extend the geodesic method of [Gong et al., 2012] and propose a deep learning model that learns the distribution shift between source and target data. They consider the path between source and target domains and learn multiple intermediate representations of the CNN features. They demonstrate promising results, but their method consists of only two layers and hence, it is significantly outperformed by deeper architectures, such as [Krizhevsky et al., 2012].

Aljundi and Tuytelaars [2016] propose a light-weight domain adaptation method, which identifies and reconstructs the filters of layers found affected by the domain shift. To spot the badly affected filters, they use a Lasso-based optimization method with a KL-D objective. In that way, they make the target filter responses more similar to the source ones. Recently, Li et al. [2016c] propose Adaptive Batch Normalization (AdaBN), a method for adjusting the statistics in all batch normalization layers. They first

concatenate the responses of each neuron for all target data, then they compute some statistics (mean and variance of the responses) for the target domain and standardize each layer as a function of this response and its statistics. In this way, they ensure that each layer receives data from a distribution that is similar for both domains.

In the following, we review methods based on encoder-decoder models and methods based on pre-trained models.

**Encoder-decoder models.** One of the first deep models for adapting sentiment classification between different domains is the Stacked Denoising Autoencoder of [Vincent et al. \[2008\]](#). Their model finds common features between the source and target domains based on denoising autoencoders. They train a CNN to reconstruct data from some randomly corrupted data by means of backpropagation. The Domain Separation Networks (DSN) [\[Bousmalis et al., 2016b\]](#) learn to extract image representations partitioned into two subspaces: a private subspace for each domain, and a subspace shared across domains. The model integrates a reconstruction loss using a shared decoder, which learns to simultaneously solve a given source task and reconstruct the input sample by using both the private (domain-specific) and source representations. The work of [Ghifary et al. \[2016\]](#) combines a standard CNN with a deconvolutional network. The model alternates between supervised training to learn the source label predictions (standard CNN), and unsupervised training to learn the unsupervised target data reconstruction (deconvolutional network). The parameters of the encoding are shared across both tasks, while the decoding parameters are kept separate.

**Pre-trained models.** Some other well known works can be seen from the domain adaptation point of view. Modern deep learning approaches use the pre-trained model of ILSVRC 2012 [\[Russakovsky et al., 2013a\]](#) as initialization and fine-tune on another target dataset. The adaptation takes place during fine-tuning with backpropagation at a lower learning rate. For instance, the R-CNN (Section 2.2.2) transfers information learned from a model pre-trained on ILSVRC 2012 to new datasets or even to new tasks (e.g., segmentation). Typically, fine-tuning works well given that the new dataset is fully annotated. Similarly, the work of [Donahue et al. \[2014\]](#) uses mid-level features pre-trained on ILSVRC 2012 to adapt to an unknown dataset. Doing so, they manage to remove the bias in some domain adaptation settings. They, however, use only one dataset and evaluate only on a subset of layers.

The method of [Hoffman et al. \[2014b\]](#) uses a pre-trained CNN for feature extrac-

tion and only trains a final domain-adapted classification layer, without any fine-tuning. They use a fully annotated source domain and only one labeled example of the target domain and show that the performance on the target domain is on par with the one on the source domain. Another work for domain adaptation is DASH-N [Nguyen et al., 2015], Domain Adaptation using a Sparse and Hierarchical Network, that jointly learns a hierarchy of features and their transformations so that they minimize the mismatch between different domains. Inspired by Bo et al. [2011], their network contains multiple layers, each one containing three sub-layers, whose output is the input of the next layer.

Instead of fine-tuning, Raj et al. [2015] use subspace alignment to adjust the feature sub-spaces between the source and the target domains. They train R-CNN (see Section 2.2.2) with labeled source images and apply it on the target domain. Then, they re-train the detector with the target aligned source features and use it to classify the target data projected into the target subspace.

### 2.3.3 Common architectures for deep domain adaption

Inspired by the siamese architectures, many works train a two-stream CNN using a standard classification loss and a combination of: (i) a discrepancy loss [Tzeng et al., 2014; Long et al., 2015, 2016a; Sun and Saenko, 2016], which aims to diminish the shift between the source and target domains, or (ii) an adversarial loss [Ganin and Lempitsky, 2015; Tzeng et al., 2015], which aims to embed the source and target features in a common feature space through an adversarial objective with respect to a domain discriminator. Figure 2.10 shows a CNN architecture for domain and task transfer. In the following, we review these two categories.

**Discrepancy-based methods.** These methods train a two-stream network relying on a discrepancy loss, usually Maximum Mean Discrepancy (MMD) [Tzeng et al., 2014]. The first work that incorporates MMD into CNNs as a confusion loss is proposed by Ghifary et al. [2014] for the fully-supervised setting of image recognition. Their two-layer network relies on MMD to reduce distribution mismatch between source and target domains, and hence manages to learn a domain invariant representation. Nevertheless, their shallow network lacks the strong semantic representation learned by deeper CNNs.

Additionally, the work of Tzeng et al. [2014] computes the MMD between the

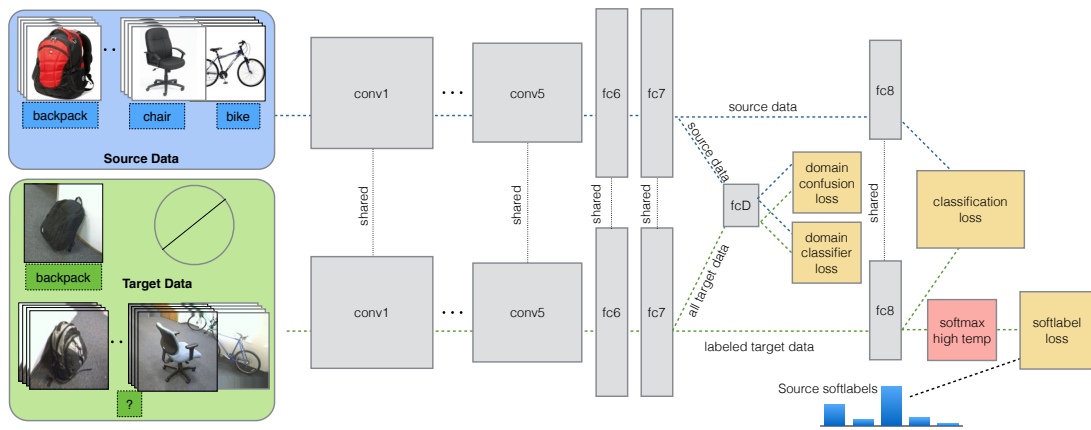


Figure 2.10: CNN architecture for domain adaptation with confusion loss and data classifier. Image source [Tzeng et al., 2015].

source and the target domain for a set of fine-tuned networks, and automatically selects the layer for discrepancy. Similarly, Long et al. [2015] propose DAN, a Deep Adaptation Network that consists of task-specific layers producing task-specific features. They embed the hidden representation of all layers into a single kernel space, where they match the domain distributions. Doing so allows them to reduce the domain discrepancy and to learn transferable features. In the same spirit, the same group proposes the joint adaptation network [Long et al., 2016a], where they consider the joint distribution discrepancy of the features from different layers instead of MMD. The recent method of Sun and Saenko [2016] presents a nonlinear transformation that aligns correlations of layers between the two streams using a CORAL loss instead of MMD.

Other works learn to adapt a CNN model from synthetic images to real world images. For instance, Rozantsev et al. [2016] consider the MMD between the weights of the source and target models of different layers, using an extra regularizing term to ensure that the weights in the two models remain linearly related.

Unlike the previous approaches, Long et al. [2016b] assume that the source and target classifiers differ by a residual factor, which is a function of the target data. Adding several layers to a network allows them to learn the residual function with reference to the target classifier. They also embed the features of different layers into kernel spaces, which allows them to match distributions of features coming from different domains. Their method can be used in other networks by adding the residual layers.

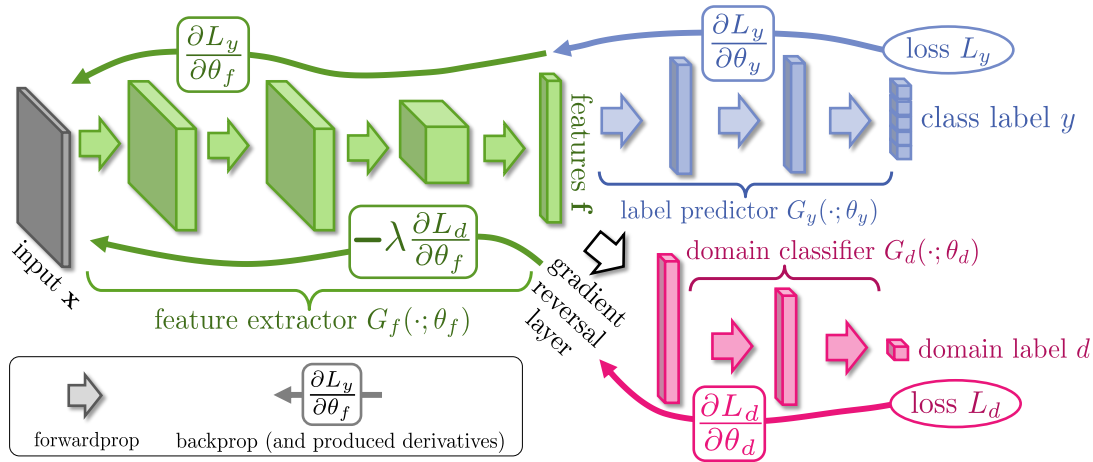


Figure 2.11: An unsupervised domain adaptation architecture. It includes a deep feature extractor (green), a deep label predictor (blue), and a domain classifier (red). Image source [Ganin and Lempitsky, 2015].

**Adversarially-based methods.** These models encourage domain confusion via an adversarial objective, i.e., not discriminate between source and target domains. The first kind of these works are the ones that optimize the domain invariance. For instance, the deep domain adaptation technique by Ganin and Lempitsky [2015] embeds domain adaptation into the feature learning process (Figure 2.11). The classification relies on the learned discriminative and domain invariant features. To do so, Ganin and Lempitsky [2015] jointly optimize the features together with two classifiers: the label predictor (to learn the training and test labels), and the domain classifier (to discriminate between source and target domains). Their method can be applied to any model by augmenting it with a gradient reversal layer that uses a minimax loss. Moreover, Tzeng et al. [2015] cast the domain invariance optimization as a joint problem of classification (predicting class labels) and of domain confusion (finding an indistinguishable representation of the data coming from the two domains). They also use their method for the task transfer problem in the semi-supervised setting.

In contrast to the above methods, the adversarial discriminative domain adaptation of Tzeng et al. [2017] considers independent source and target mappings (no shared weights between the two streams). They train a network on the source domain and use it to initialize the target network, allowing the target network to learn domain specific features. They use an inverted label GAN loss [Goodfellow et al., 2014] to split the optimization into two independent objectives, one for the generator and one for the discriminator. A Generative Adversarial Network (GAN) [Goodfellow et al., 2014] is



a system of two CNNs competing against each other in a zero-sum game framework. One network is generative and one is discriminative. Briefly, the discriminator discriminates between instances from a true data distribution and synthesized instances produced by the generator. The generator wants to *fool* the discriminator by producing novel synthesized instances that appear to have come from the true data distribution.

Apart from [Tzeng et al., 2017] there are also some other methods that combine the discriminative model with a generative model relying on GANs [Goodfellow et al., 2014]. For instance, the Coupled Generative Adversarial Networks (CoGAN) of Liu and Tuzel [2016] learn a joint distribution of multi-domain images (RGB and depth) from just samples drawn from the marginal distributions of images from both domains. CoGAN enforces a weight sharing constraint to limit the network capacity. In the same spirit, Bousmalis et al. [2016a] use GANs to generate source images that appear like they are drawn from the target distribution.

### 2.3.4 Heterogeneous domain adaptation

Here, we review adaptation methods where the modality of the two domains is different. This is very common in real-world scenarios, where we want to generalize models from images to videos [Donahue et al., 2013; Gaidon and Vig, 2015; Sharma and Nevatia, 2013; Tang et al., 2012], optical flow [Gupta et al., 2016], depth images [Chen et al., 2014a; Gupta et al., 2014, 2015; Hoffman et al., 2016; Liu and Tuzel, 2016; Song and Xiao, 2014; Wang et al., 2015a], etc. For instance, the work of Gupta et al. [2014] presents a geocentric embedding for depth images that encodes features of the depth images, such as height or disparity. Their method uses RGB and depth image pairs to learn a feature representation used for object classification. They show that the proposed embedding works better than using raw depth images. Depth images are also exploited by Song and Xiao [2014], who learn rich features from depth images, and by Gupta et al. [2015], who train a CNN on depth images to infer the pose of an object. Additionally, the work of Wang et al. [2015a] presents a CNN multi-modal framework, where they first train on RGB and depth images separately, and then connect them with a multi-modal layer. This layer discovers the most discriminative features for each modality, while exploiting the complementary relationship between them.

The recent work of Gupta et al. [2016] learns representations for pairs of images from different modalities, where one modality is labeled (typically RGB images), whereas the other one is unlabeled (depth images or optical flow). They transfer su-



pervision to the unknown modality by teaching the network to reproduce the mid-level semantic representations of the RGB images.

Other works learn to adapt a CNN model from synthetic images to real world images [Rozantsev et al., 2016; Massa et al., 2016], or even use different non-vision related modalities, such as combinations of images or videos with text or audio [Ngiam et al., 2011; Chakravarty and Tuytelaars, 2016], showing that learning representations from multiple modalities, such as video and audio, outperforms learning from a single modality.

Additionally, the Transfer Neural Trees by Chen et al. [2016] relate heterogeneous cross-domain data. They use a two-stream network with shared weights between the endings of the two streams, as well as a Transfer Neural Decision Forest (Transfer-NDF) that jointly performs adaptation and classification to predict the class labels.

Some other approaches operate in the weakly-supervised setting. For example, the weakly-shared deep transfer network by Shu et al. [2015] learns a domain translator function from multi-modal source data. This function predicts the target labels even if only one modality is present. Their method is able to represent both domain-specific features and shared features across domains. Another weakly-supervised method is proposed by Li et al. [2016a], who work on progressive domain adaptation. They address the task in two steps: classification adaptation and detection adaptation. In the first step, they transfer a pre-trained network to a multi-label classification task, aiming at removing background clutter and potential confusion from similar objects. In the second step, they fine-tune the network with class-specific object proposals that they have collected with high confidence from their trained network.

Another well known heterogeneous domain adaptation scheme is adapting from images to videos, or vice versa for object detection. The early works adapt a detector trained on labeled source images to detect objects in videos, typically cars or pedestrians. Most of these works [Gaidon et al., 2013, 2014; Gaidon and Vig, 2015; Sharma and Nevatia, 2013; Tang et al., 2012] combine the generic detector trained on images with object tracking to obtain target positive samples. In Chapter 3.2 we give a more extensive review of works adapting detectors from videos to images and vice versa.

## 2.4 Action localization

Action localization (also called action detection) is one of the core challenges for better understanding videos. The goal of action localization is to identify where, when,

or both an action takes place in a video. There are two types of action localization: temporal [Duchenne et al., 2009; Gaidon et al., 2013; Niebles et al., 2010; Yuan et al., 2009] and spatio-temporal [Cao et al., 2010; Gkioxari and Malik, 2015; Jain et al., 2014; Klaser et al., 2010; Lan et al., 2011; Laptev and Pérez, 2007; Peng and Schmid, 2016; Saha et al., 2016; Tran and Yuan, 2011, 2012; Wang et al., 2014; Weinzaepfel et al., 2015]. In temporal localization the goal is to find the temporal span of an action, i.e. the beginning and ending frames. Spatio-temporal localization, as its name suggests, finds where an action occurs in space *and* in time, i.e. temporally finding the beginning and ending frames of the action, and spatially finding the bounding boxes that cover the action.

Some action localization techniques focus on finding only the temporal extent of actions in videos [Caba Heilbron et al., 2016; Escorcia et al., 2016; Duchenne et al., 2009; Oneata et al., 2014b], while most techniques focus on the actions of humans in space and time [Gkioxari and Malik, 2015; Peng and Schmid, 2016; Saha et al., 2016]. Note that there are no works that focus only on the spatial localization of humans, as the datasets for action localization provide also temporal information. There are, however, some very recent works that focus on detecting moving objects in videos only spatially [Kang et al., 2016b,a, 2017; Zhu et al., 2017b,a]. The temporal extent of the moving objects is not known, and hence these works do not detect the objects temporally. Even though these works focus on objects instead of humans, the methodology they use for detection imitates the one from action localization.

**Outline.** We first review techniques for temporal localization in Section 2.4.1, and then for spatio-temporal localization in Section 2.4.2. Next, in Section 2.4.3 we introduce another category of works that resembles the action localization methods: these works detect moving objects in videos (spatial localization) leveraging the temporal information. Finally, Section 2.4.4 describes the modern action localization datasets used in our experiments.

### 2.4.1 Temporal action localization

The most common technique for temporal action localization is sliding window, casting the localization as localized classification [Escorcia et al., 2016; Caba Heilbron et al., 2016]. Duchenne et al. [2009] first apply a classifier to uniformly extract temporal windows of interest, i.e., temporal windows likely to contain the action of interest.

Then, they compose a score for each window to finally keep the one with the highest score. As the search space is only on the temporal dimension, the complexity of these models is within the computational limitations.

There are, however, some works that improve upon the search complexity [Gaidon et al., 2013; Oneata et al., 2014b; Yuan et al., 2009]. Gaidon et al. [2013] propose a more structured model for temporal action localization. They model each action as three consecutive sub-actions, which are represented with the BOW encoding. This model allows learning the duration of each sub-action in a non-parametric way, but requires additional annotations for each sub-action. Niebles et al. [2010] apply the DPM detector to the temporal domain: they infer temporal anchor points and scales for the sub-events of each action class. Then, they pool the features from the temporal locations and compare them. Oneata et al. [2014b] reduce the complexity and the computational cost of temporal action localization. Relying on pre-computed sums of local BOW, scores, and Euclidean norms, they propose an approximately normalized FV method. Inspired by the branch-and-bound search method of Lampert et al. [2009], their method scores any arbitrarily-sized window in constant time, in contrast to the sliding window techniques that require a large number of windows to be evaluated.

Some action localization works operate in the *weakly-supervised* setting [Bojanowski et al., 2014; Duchenne et al., 2009; Hoai et al., 2014; Satkin and Hebert, 2010]. In these works the training annotations are noisy: there is no accurate annotation but only a rough estimation of the beginning and ending of the actions. For instance, some works use discriminative clustering either to refine the localization of actions [Duchenne et al., 2009], or to assign temporal segments to actions [Bojanowski et al., 2014]. The former learns classifiers for each action while the latter learns a detector. Classifiers, and in particular multiple instance SVMs, are also used by Hoai et al. [2014], who extend the classifiers to time series, allowing discontinuities in the positive samples. Finally, to find discriminative segments in videos, Satkin and Hebert [2010] use a max-margin objective function with temporal extents acting as latent variables.

## 2.4.2 Spatio-temporal action localization

As in temporal action localization, the most straightforward technique for spatio-temporal action localization is extending the sliding window paradigm to space and time [Cao et al., 2010]. In this case, however, the search space is too large to efficiently compute scores for each sliding window. In an attempt to reduce the search space, the first

approaches to spatio-temporal action localization required strong assumptions, for instance they assumed a fixed spatial extent of the action [Cao et al., 2010; Laptev and Pérez, 2007]. These assumptions were abandoned by later techniques, as most of these assumptions are not realistic for real-world videos, where the actors and the camera move. Other sliding window approaches extend the DPM detector (Section 2.2.1) to videos. For instance, the work of Tian et al. [2013] extends the DPM to the spatio-temporal domain by replacing the HOG filters with their 3D version, i.e., the HOG-3D filters of [Klaser et al., 2008].

The HOG-3D filters are also used by Klaser et al. [2010] to describe and classify human tracks for action localization. They produce these tracks using KLT feature tracks of human detectors. Inspired by the latent SVM framework of DPM, Lan et al. [2011] cast the action localization problem as a set of image-level detection problems: they consider the location of a person in each frame as a latent variable and the deformation cost enforces similar locations across time.

Other action localization works require more annotations, such as pose or other objects. For instance, Prest et al. [2013] detect humans and objects and then model their interaction, while Jhuang et al. [2013b] leverage pose to improve action localization. The former approach needs extra object annotations, while the latter requires pose annotations. Additional pose annotations are used by Wang et al. [2014], who use a temporal sliding window and then model the relations between the so called dynamic poselets, which correspond to some key short sequences of pose configuration.

We divide the rest of the action localization works into three categories: approaches based on action proposals, per-frame approaches based on object detectors, and tubelet approaches based on object detectors. Below, we provide more details about them.

**Methods based on action proposals.** Object proposals are regions within an image likely to contain an object of interest. They were first introduced by Alexe et al. [2010] by defining an objectness score for multiple regions in an image. Since then, there have been numerous works for generating fast and accurate proposals [Manen et al., 2013; Zitnick and Dollár, 2014] and their usage is well-established in modern computer vision methods. The most used proposal generator is the Selective Search method (SS) of Uijlings et al. [2013]. For instance, the work of Marian Puscas et al. [2015] uses object proposals generated by Selective Search and performs action localization by linking the dense trajectories [Wang et al., 2013] of these proposals throughout the video.

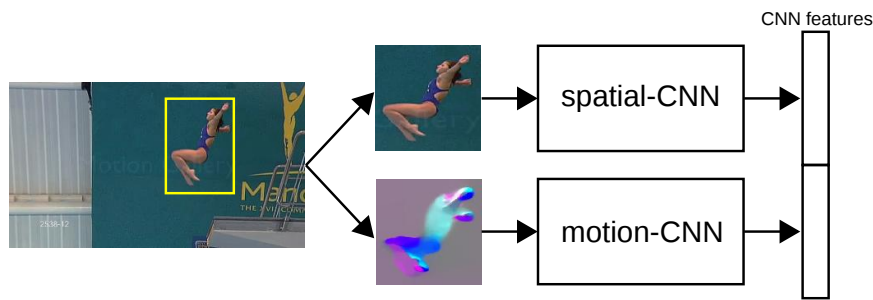


Figure 2.12: Example of a modern two-stream CNN architecture for predicting action labels. Image source [Weinzaepfel et al., 2015].

Inspired by object proposals, there are also a few other methods that generate action proposals, i.e., spatio-temporal windows of interest in a video. These works cast action localization as a proposal classification problem. For instance, Jain et al. [2014] extend Selective Search to videos. Starting from a video segmentation [Xu and Corso, 2012], they greedily construct action proposals by hierarchically merging supervoxels. They finally classify these action proposals and perform spatio-temporal localization in an efficient manner. Similarly, Oneata et al. [2014a] extend the approach of [Manen et al., 2013] to the video domain, by using supervoxels as units to merge into video proposals. Gemert et al. [2015] generate proposals by efficiently clustering dense trajectory features, thus avoiding the video segmentation step of [Jain et al., 2014; Oneata et al., 2014a]. Then, they classify the tubes based on dense trajectory features and perform action localization efficiently.

Similarly to the objectness measurement, some more recent works [Chen et al., 2014b; Li et al., 2016d; Wang et al., 2016a; Yu and Yuan, 2015] rely on the actionness measurement [Chen et al., 2014b], i.e., a pixel-wise probability of containing any action. For instance, Yu and Yuan [2015] estimate actionness using optical flow, and extract action tubes by solving a maximum set coverage problem. Similarly, Wang et al. [2016a] estimate actionness using hybrid fully-convolutional CNNs leveraging static appearance and dynamic motion, and perform action localization by classifying the proposals. Li et al. [2016d] introduce videoLSTM, which applies attention in convolutional LSTM models to discover relevant spatial-temporal volumes. These works, however, output only a rough localization of the action as the localization is based on noisy pixel-level maps.

**Per-frame approaches based on object detectors.** The majority of modern methods for action localization rely on object detectors trained on humans at the frame level (Figure 2.12). [Gkioxari and Malik \[2015\]](#) extend the R-CNN framework to a two-stream variant [[Simonyan and Zisserman, 2014](#)] processing RGB and flow data separately. The first stream (RGB) focuses on appearance features, while the second one (flow) focuses on flow features, given that motion is a relevant cue for action detection. Object proposals from Selective Search are detected and classified in each frame using R-CNN. Then, these detections are linked using dynamic programming with a cost function based on detection scores of the boxes and overlap between detections of consecutive frames. [Weinzaepfel et al. \[2015\]](#) obtain tubes using a tracking-by-detection method instead of linking per-frame detections, as tracking-by-detection is more robust, for instance in the case of multiple actors.

More recently, the two-stream network style was also used with Fast R-CNN, Faster R-CNN, and SSD detectors [[Peng and Schmid, 2016](#); [Saha et al., 2016](#); [Singh et al., 2017](#)]. The method of [Saha et al. \[2016\]](#) trains two RPNs of the Fast R-CNN network, one with RGB frames and one with optical flow, to generate a set of region proposals with their associated scores. Then, they train Fast R-CNN with the generated proposals and video frames, resulting in regressed detections with classification scores. They fuse the softmax scores of both streams based on the overlap between the appearance and the motion RPNs. Finally, they generate tubes for each action class, based on the scores and the spatial overlap between consecutive detections, that they then trim over time. Similarly, [Peng and Schmid \[2016\]](#) extract Selective Search proposals from both streams that they combine after the *conv5* layer. Each proposal is represented with multiple regions, helping the detection of human parts, such as legs, shoulders, etc. They train each stream with the combined proposals, and then classify and regress them with fused RGB and multi-frame optical flow features. Finally, they link the regressed proposals across a video based on their spatial overlap and coherence of their classification scores. The last work that uses a two-stream architecture is presented by [Singh et al. \[2017\]](#), who perform real-time action localization using the efficient SSD detector and the fast method of [Kroeger et al. \[2016\]](#) to estimate the optical flow for the motion stream. Then, they introduce a greedy online linking algorithm that incrementally builds tubes for each action class.

[Zolfaghari et al. \[2017\]](#) extend the two-stream architecture to a three-stream one, adding a stream for human parts. They claim that the additional stream can capture temporal dynamics of body parts over time. For combining the three streams, instead

of fusing their scores, they integrate them via a Markov chain, which refines the action labels. To train the pose network, they use the Fast-Net [Oliveira et al., 2016], which provides them with segmentations of human body parts.

**Tubelet approaches based on object detectors.** Most approaches for action localization work on per-frame detections. Very recently the community started shifting towards sequences of frames for action localization [Hou et al., 2017; Saha et al., 2017].

For instance, the work of Hou et al. [2017], called Tube Convolutional Neural Network (T-CNN), uses clips of fixed length (eight frames) to generate tube proposals based on 3D CNN features. These tube proposals are linked over times based on their score and the spatial overlap of consecutive proposals, and form tubes. Inspired by the RoI of Fast R-CNN [Girshick, 2015a] (Section 2.2.3), they also introduce a Tube-of-Interest (ToI) pooling layer, that is applied to the linked action tube proposals to generate fixed-length feature vectors.

In contrast, Saha et al. [2017] build their Action-Micro-Tube network (AMTnet) on only two consecutive frames. AMTnet is an end-to-end parallel network where each stream takes as input one frame, and their two output feature maps are fused and passed as input to a 3D Region Proposal Network (3D-RPN). The 3D-RPN generates micro-tube proposals, i.e., a pair of bounding boxes, with their associated scores. Then, they apply bilinear feature pooling to each box independently, resulting in two pooled feature maps to which they apply element-wise fusion. The final feature maps are fed to the *fc6* and *fc7* layers of the network, which output regressed micro-tubes with scores.

In Chapter 5, we present our ACTION Tubelet detector (ACT-detector) that captures all the advantages of the related work: it is an end-to-end two-stream detector (working on appearance and motion cues), trained with sequences of frames instead of single frames, that outputs regressed tubelets with associated scores.

### 2.4.3 Spatial localization for video object detection

The goal of video object detection is to detect objects only spatially, as the temporal extent of the objects appearing in the videos is unknown, and therefore no temporal detector can be trained. Only during the last couple of years (2016-2017), some works started exploiting the temporal continuity of moving objects [Kang et al., 2016b,a,



2017; Zhu et al., 2017b,a] and the methodology they follow resembles the one from action localization. In line with modern action localization techniques, these approaches use two-stream architectures, leveraging the appearance and the motion of the objects to improve the spatial localization of objects.

The first work of this kind was introduced by Kang et al. [2016b], who propose a tubelet proposal framework, where objects are tracked in time. They generate new tubelet proposals by applying tracking algorithms to per-frame proposals. The scores of the boxes in a tubelet are evaluated with an object detector before being re-scored using another CNN architecture. The same group improved on their tubelet classification and re-scoring strategy, resulting in T-CNN [Kang et al., 2016a], which won the ILSVRC 2015 VID challenge [Russakovsky et al., 2015b] with provided data. Given an initial set of per-frame detections, T-CNN propagates these predictions to neighboring frames according to the precomputed optical flow between frames, and then generates tubelets by applying tracking algorithms to bounding boxes with high score. The boxes of the tubelets are then re-scored based on classification of the tubelets: the class scores of boxes not belonging to the top classes are removed to enforce temporal consistency of class scores.

More recently, inspired by the RPN of object detectors, the same group proposed a Tubelet Proposal Network (TPN) [Kang et al., 2017]. TPN consists of two sub-networks: one extracts visual features across time based on per-frame proposals, and the other is a regression layer that estimates displacements of the bounding boxes across time. TPN relies on the large receptive field of CNNs, and performs feature map pooling at all locations across time of each bounding box to extract the visual features of moving objects. As in TPN, the deep feature flow framework for video recognition of Zhu et al. [2017b] also consists of two sub-networks: one extracts features using ResNet [He et al., 2016] for classification, and the other computes the optical flow using FlowNet [Dosovitskiy et al., 2015]. The key element is that the first sub-network runs only on key-frames (sparsely extracted), and then the feature maps of the non-key-frames are propagated from the key-frames to the second flow sub-network. Another very recent work of this kind is introduced by Zhu et al. [2017a], who present an end-to-end network with flow-guided feature aggregation for video object detection based on the R-FCN network [Li et al., 2016b]. Given a reference frame, they use FlowNet [Dosovitskiy et al., 2015] to compute the flow between this frame and a neighboring one, and then they wrap the feature maps of the neighboring frame to the reference one. In that way the reference frame has multiple feature maps, that



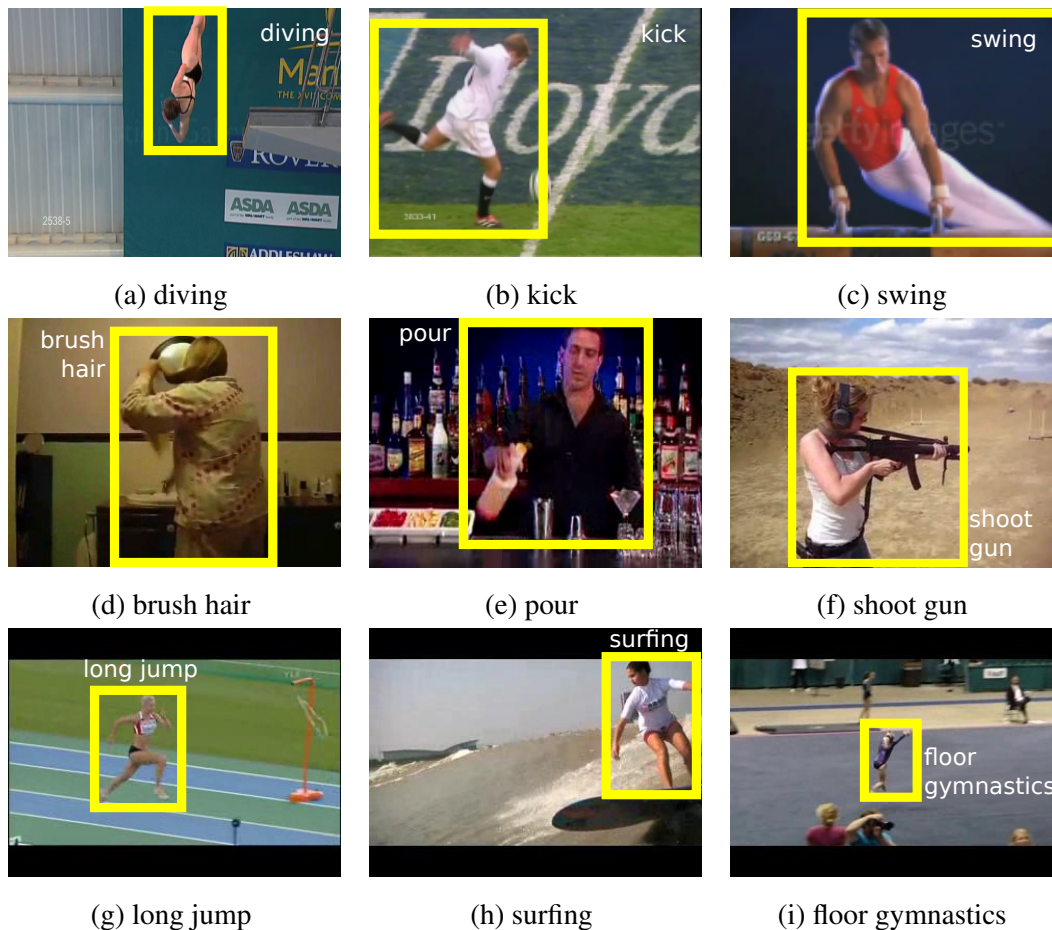


Figure 2.13: From top to bottom, example frames from the UCF-Sports, J-HMDB, and UCF-101 action localization datasets.

they project in the same feature space with an embedding sub-network. Finally, they aggregate the feature maps derived from the embedding network with a weighted sum.

All the aforementioned methods are applied only in video object detection datasets: ILSVRC VID [Russakovsky et al., 2015a] and YouTube-Objects [Kalogeiton et al., 2016; Prest et al., 2012a]. These datasets do not contain any information about the first or last frame in which an object appears. Therefore, in contrast to action localization techniques, these methods are tested only on the spatial domain.

#### 2.4.4 Action localization datasets

In this section, we describe the three most common action localization datasets: UCF-Sports, J-HMDB, and UCF-101. Table 2.1 reports some statistics about the datasets and Figure 2.13 illustrates some examples. These are used in our experiments in Chapters 4-5.

| information / datasets | UCF-Sports       | J-HMDB           | UCF-101          |
|------------------------|------------------|------------------|------------------|
| action types           | sports           | everyday         | sports           |
| # classes              | 10               | 21               | 24               |
| # videos               | 150              | 928              | 3207             |
| # frames               | 10k              | 32k              | 558k             |
| # instances            | 154              | 928              | 4030             |
| avg resolution         | $690 \times 450$ | $320 \times 240$ | $320 \times 240$ |
| avg video duration     | 5.8s             | 1.4s             | 5.8s             |
| avg action duration    | 5.8s             | 1.4s             | 4.5s             |

Table 2.1: Overview of the action localization datasets used in our experiments.

The **UCF-Sports** dataset [Rodriguez et al., 2008] consists of 150 videos from 10 sports classes such as *diving*, *running*, *skateboarding*, etc. All videos are trimmed to the action duration and every frame is annotated with a bounding box. The average duration of the videos is 6 seconds. In our experiments, we use the train/test split defined by Lan et al. [2011], where there are 9.5k bounding boxes, with almost 3k test ones.

The **J-HMDB** dataset [Jhuang et al., 2013a] contains 928 videos with 21 actions, including *brush hair*, *climb stairs*, *shoot gun*, etc. It is a subset of the HMDB dataset [Kuehne et al., 2011] for action classification. The videos are short with an average duration of 1.4 seconds and are trimmed to the action duration. We report results averaged on the three splits defined in [Jhuang et al., 2013a], unless stated otherwise. In total there are around 32k frames annotated with bounding boxes, with approximately one third (a bit less than 10k) belonging to the test set.

The **UCF-101** dataset [Soomro et al., 2012] contains more than 13k videos for 101 classes for action classification. For a subset of 24 classes, there are also spatio-temporal annotations, resulting in 3207 videos. It contains classes such as *horse riding*, *soccer juggling*, *tennis swing*, etc. The videos last on average 6 seconds. In contrast to the previous datasets, the videos are not trimmed; the action duration, however, covers a significant part of them. In total there are around 558k annotated frames, with three train and test splits. Following all recent works on action localization [Gkioxari and Malik, 2015; Peng and Schmid, 2016; Saha et al., 2016; Weinzaepfel et al., 2015], when using UCF-101 we report results for the first split only.



# Chapter 3

## Analyzing domain shift factors between images and videos for object detection

### Contents

---

|            |  |           |
|------------|--|-----------|
| <b>3.1</b> | <b>Overview of our approach</b>                                  | <b>57</b> |
| <b>3.2</b> | <b>Related work</b>  | <b>59</b> |
| 3.2.1      | Adapting object detectors from videos to images                  | 59        |
| <b>3.3</b> | <b>Datasets and protocol</b>                                     | <b>62</b> |
| 3.3.1      | First dataset pair   | 62        |
| 3.3.2      | Second dataset pair  | 64        |
| 3.3.3      | Protocol   | 64        |
| <b>3.4</b> | <b>Analysis of domain shift factors and experimental results</b> | <b>66</b> |
| 3.4.1      | Spatial location accuracy  | 67        |
| 3.4.2      | Appearance diversity   | 71        |
| 3.4.3      | Image quality  | 73        |
| 3.4.4      | Aspect distribution  | 76        |
| 3.4.5      | Other factors  | 80        |
| 3.4.6      | Experiments on the ILSVRC 2015 dataset pair                      | 81        |
| <b>3.5</b> | <b>Conclusions</b>   | <b>83</b> |

---

Object class detection is a central problem in computer vision. Training an object detector is usually done with still images. Recently, videos have been used as an alternative rich source of training data. As opposed to still images, video provides several advantages:

1. the motion enables to automatically segment the object from the background [Papazoglou and Ferrari, 2013], replacing the need for manually drawing bounding-boxes;
2. a single video often shows multiple views of an object; and
3. videos contain multiple deformation and articulation states (e.g. for animal classes).

Recent works [Kim et al., 2014; Leistner et al., 2011; Prest et al., 2012a; Sharma and Nevatia, 2013; Tang et al., 2012, 2013] started to exploit both sources of data for object detection, by transferring information extracted from the video domain to the still images domain, or vice versa. These works operate in a domain adaptation setting [Pan and Yang, 2010; Gopalan et al., 2014; Duan et al., 2012c] and show that when testing on a target domain, there is a significant performance gap between training on this domain or on a different one. This is due to the different nature of the two domains.

In this chapter, we explore the differences between still images and video frames for training and testing an object detector. We consider several domain shift factors that make still images different from video frames. We are the first to analyze with a structured protocol such domain shift factors to reveal the source of the performance gap. Figure 3.1 shows the pipeline we follow for our exploration. This work is published in [Kalogeiton et al., 2016]. As part of this work, we also released the YouTube-Object v2.0-v2.3 dataset<sup>1</sup>, which are available at <http://calvin.inf.ed.ac.uk/datasets/youtube-objects-dataset/> and <https://github.com/vkalogeiton/yto-dataset>.

**Outline.** We first present an overview of our approach in Section 3.1. Then, we review the related work in more details in Section 3.2, and in Section 3.3 we introduce the datasets and protocol we work on. In Section 3.4 we present with more details our work for analyzing domain shift factors for object detection together with extensive experimental results, and we conclude in Section 3.5. In Appendix B we report per-class

---

<sup>1</sup>[Dataset viewer](#)

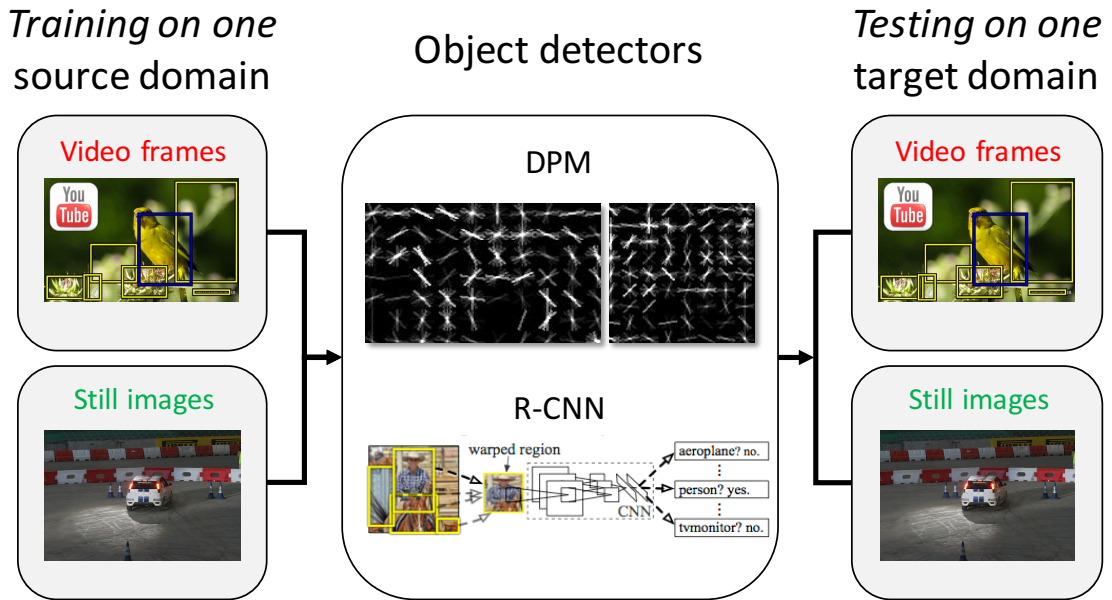


Figure 3.1: Our pipeline for examining the impact of domain shift factors between videos and images for object detection. First, we train two object detectors (DPM and R-CNN) on one source domain (videos or images). Then, we test on both domains and examine the difference in the detection performance.

results for all the experiments of this chapter. In Appendix C we present additional experiments when training object detectors from both still images and video frames.

### 3.1 Overview of our approach

Training an object class detector on one domain (either still images or videos) and testing on the other one results in a significant performance gap compared to training and testing on the same domain. In this chapter, we explain the reasons behind this performance gap as several domain shift factors that make the nature of still images different than the one of video frames.

We carry out our investigation on four datasets grouped into two pairs. Each pair contains an image and a video dataset (Section 3.3). The first pair is PASCAL VOC 2007 of [Everingham et al. \[2007\]](#) (images) and YouTube-Objects<sup>2</sup> of [\[Prest et al., 2012a; Kalogeiton et al., 2016\]](#) (video). Both datasets in the second pair come from ILSVRC 2015 [\[Russakovsky et al., 2015a,b\]](#), i.e., from the ‘object detection in images’

<sup>2</sup>As part of this work, we released the v2.0-v2.3 of the dataset, where we annotated almost 7k bounding boxes. The dataset is available at <http://calvin.inf.ed.ac.uk/datasets/youtube-objects-dataset/> and <https://github.com/vkalogeiton/ytto-dataset>.

and ‘object detection in videos’ tracks of the challenge.

We identify and analyse five kinds of domain shift factors that make still images different from video frames (Section 3.4).

The first is the *spatial location accuracy* of the training samples (Section 3.4.1). As most previous experiments on training detectors from videos were done in a weakly supervised setting, we quantify how much of the performance gap is due to the poor quality of automatically generated bounding-boxes. In contrast, still image detectors are typically trained from manually drawn bounding-boxes.

The second factor we consider is the *appearance diversity* of the training samples within a domain (Section 3.4.2). Video differs from still images in that frames are temporally correlated. Frames close in time often contain near identical samples of the same object, whereas in still image datasets such repetition happens rarely. This is an intrinsic difference in the medium, which is often overlooked.

The third factor is *image quality* (Section 3.4.3), which includes level of blur, color accuracy and contrast, radial distortion, exposure range, compression artifacts, lighting conditions, etc. In this work, we consider Gaussian blur and motion blur, since we empirically found that level of blur is one of the most important differences between videos and still images.

The fourth factor is the *distribution over aspects*. The *aspect* of a sample is the type of object appearing in a training set. For example, the object *horse* can have various aspects: *horse’s head*, *horse’s head and neck*, *horse jumping over hurdles* etc. (Section 3.4.4). As the space of possible aspects for an object class is very large, each dataset covers it only partially [Torralba and Efros, 2011], with its own specific bias. For example, a *horse jumping over hurdles* might appear in one dataset but not in another. Hence, an important factor is the differences in the aspect distributions between the two domains.

As last factor, we consider *object size and camera framing* issues (Section 3.4.5). Photographers and videographers might follow different approaches when capturing an object, e.g., in images the objects tend to be fully in focus, while videos might have objects coming in and out of the frame. Also the distance at which objects are captured might be different. Hence, we consider the distribution of object size, aspect-ratio, and truncation by the image frame as a last factor.

We proceed by examining and evaluating each domain shift factor in turn, following the same structure:

- we introduce a metric to quantify the factor in each domain (*Measurement*);

- we cancel out the factor by modifying the training set of each domain so that the modified training sets are more similar in terms of this metric (*Equalization*);
- we examine the impact of this equalization on the performance of the object detector (*Impact*).

As we found no difference in the last factor, i.e. object size and camera framing between the two datasets (Section 3.4.5), we report this procedure only for the first four factors.

We consider the *performance gap*, i.e. the difference in performance of a detector trained on video frames or on still images (on a fixed test set consisting of *still images*). We examine the evolution of the performance gap as the training sets get progressively equalized by the procedure above. We also repeat the study in the reverse direction, i.e. where the test set is fixed to video frames (Figure 3.1). The results show that all factors affect detection performance and that cancelling them out helps bridging the performance gap.

We perform experiments on two popular object detection models, DPM of Felzenszwalb et al. [2010] and R-CNN of Girshick et al. [2014a], see Sections 2.2.1–2.2.2. While these are very different, our results hold for both, suggesting that our findings apply to object detection in general. Moreover, the results follow the same trends on both dataset pairs we considered, showing that the domain shift factors we examine are relevant in general, and the effects we observe are not specific for some image or video datasets but can be generalized to most modern computer vision datasets.

## 3.2 Related work

Traditionally, object detectors are trained from still images and aim at detecting objects in images (Section 2.2). As we discuss in Section 2.3.2, there are several approaches for object detection that operate under the domain adaptation setting: transfer information from videos (source domain) to images (target domain) and vice versa. Here, we give an overview of these two categories.

### 3.2.1 Adapting object detectors from videos to images

Several approaches exist, in which the source domain is video and the target domain is still images [All et al., 2011; Hartmann et al., 2012; Leistner et al., 2011; Liang et al., 2015; Prest et al., 2012a; Tang et al., 2013].



[Leistner et al. \[2011\]](#) use unlabeled video data to improve the performance of classifiers for detection on images. They detect moving objects in a video and use them to learn a part-based random forest detector on still images. A different approach for detecting objects is presented in [\[All et al., 2011\]](#). They propose FlowBoost, a model trained with video datasets where only a part of the videos has an object class label. The training procedure alternates between training the object detector based on appearance, and training a time-based regularization that relabels the video. Moreover, [Liang et al. \[2015\]](#) detect objects on images by training an object detector from both images and videos that are mined from video websites, e.g., YouTube. Initially, they train a detector from very few positive image instances and test it on videos. Then, they track detected video instances and augment the training set with more relevant (in terms of appearance and spatial correspondence) video instances.

[Prest et al. \[2012a\]](#), [Tang et al. \[2013\]](#), and [Hartmann et al. \[2012\]](#) present weakly-supervised techniques for automatically annotating spatio-temporal segments on objects in videos tagged as containing a given class. The weakly-supervised algorithm of [Tang et al. \[2013\]](#) annotates spatio-temporal segments using video-level tags provided by Internet videos. Their goal is to generate video data suitable for training object detectors for image challenges. The method of [\[Hartmann et al., 2012\]](#) trains object models solely from weakly-tagged internet videos. For a given video, they extract spatio-temporal segments that they classify for each object category. Finally, they use these segments to detect objects at the pixel level. The technique of [Prest et al. \[2012a\]](#) aims at learning object class detectors from video data. They use weakly-annotated videos, and they automatically produce spatio-temporal bounding-boxes (tubes) of the objects in these videos. These are then used to train object detectors. Their experiments, however, show that training object detectors on still images outperforms training on video frames.

**Adapting object detectors from images or videos to videos.** There are only a few works [[Cherniavsky et al., 2010](#); [Sharma et al., 2012b](#); [Sharma and Nevatia, 2013](#); [Tang et al., 2012](#); [Wang et al., 2012](#)] that adapt object detectors to the video domain. These works typically deal with a constrained set of videos and limited object classes.

Some of these works use videos as both source and target domain. For instance, both the methods of [[Misra et al., 2015](#); [Pirsiavash et al., 2011](#)] use videos to detect objects in videos. In both cases, the domain adaptation part lies in using still images as negative training data. The algorithm of [[Misra et al., 2015](#)], for instance, starts with

annotated videos, and iteratively discovers new instances from a pool of unlabeled videos. Similarly, Wang et al. [2012] propose a non-parametric detector adaptation method. First, they train offline an object detector on videos, and then they adapt it to the visual characteristic of a specific video clip.

Other works such as [Sharma and Nevatia, 2013; Tang et al., 2012] use still images as source domain and video frames as target domain. For instance, Sharma and Nevatia [2013] propose an online adaptation method, which adapts a detector trained offline on images to a test video. They show that the performance of the detector on videos can be significantly improved by this adaptation, as the initial image training samples and the video test samples can be very different. In the same spirit, Donahue et al. [2013] adapt object detectors trained on static images to the video domain. They adapt an object detector (DPM, see Section 2.2.1) trained on the source image domain [Everingham et al., 2007] to the target video domain, by exploiting the signal in the temporal structure of the video data (that they automatically extract using [Lee et al., 2011]) and hence, by imposing similarity constraints on the adapted detector. Moreover, Gaidon and Vig [2015] adapt object detectors online for multi-object tracking. They jointly learn target models by adapting them from the pre-trained one, which is adapted online. Finally, Tang et al. [2012] introduce a self-paced domain adaptation algorithm to iteratively adapt an object detector from labeled images to unlabeled videos.

Also inspired by self-paced learning<sup>3</sup>, Xu et al. [2014] automatically label samples in the video target domain. Their unsupervised approach adapts online the DPM detector (Section 2.2.1) for pedestrian detection in videos. They exploit the temporal continuity of videos by using similarity constraints on the adapted detector. In contrast to self-paced learning, the unsupervised online self-learning method of Gaidon et al. [2014] considers no access to the target data. Their method aims at learning detectors to continuously update themselves adapting to each new stream of data using a multi-task learning convex objective. Another unsupervised approach is presented by Sharma et al. [2012b]. They propose an unsupervised multiple instance learning (MIL) method to improve the performance of a detector trained offline for a given video sequence. The approach of Cherniavsky et al. [2010] presents a weakly-supervised setting to learn facial attributes of humans in videos. They use a small set of images labeled

---

<sup>3</sup>In human education, self-paced learning refers to a system where the curriculum is determined by the pupils abilities rather than being fixed by a teacher. In computer vision, self-paced models [Kumar et al., 2010] typically simultaneously select *easy* samples and update the parameters at each iteration. The *easy* samples are defined as a set of samples (not individual ones) with the highest confidence; a set of samples is easy if it admits a good fit in the model space.

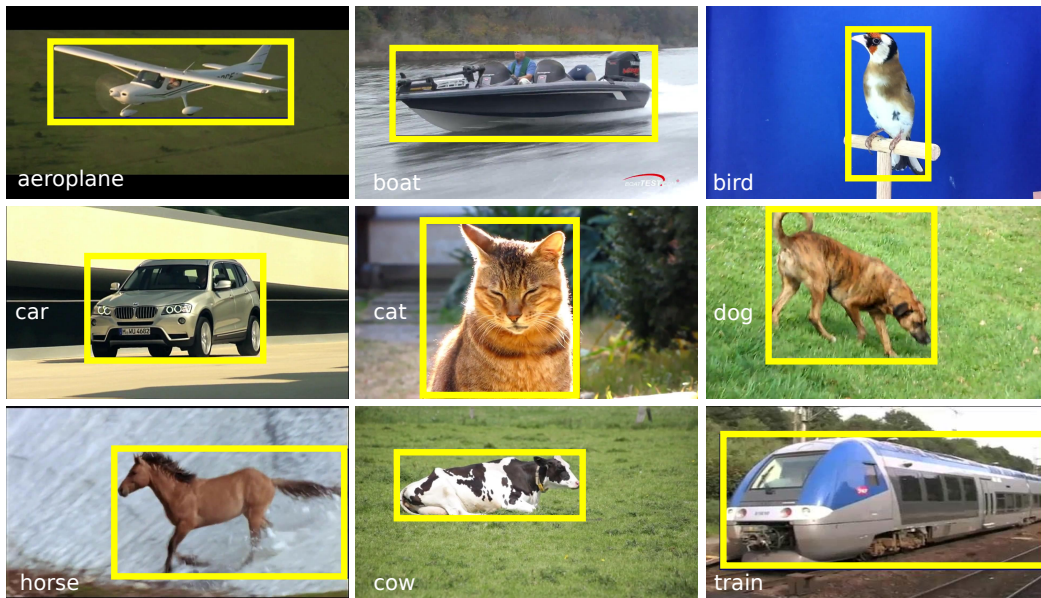


Figure 3.2: Example YTO frames with ground-truth bounding-boxes.

with attributes to train a classifier. Then, they use this classifier to iteratively recognize attributes in unlabeled videos.

### 3.3 Datasets and protocol

We perform experiments in image and video datasets. To this end, we use two dataset pairs: (a) PASCAL VOC 2007 and YouTube-Objects and (b) ImageNet Large Scale Visual Recognition Challenge (ILSVRC [Russakovsky et al., 2015b]) object detection in images (IMG) and in videos (VID) tracks of the challenge. We train two modern object detectors: DPM [Girshick et al., 2012b], and R-CNN [Girshick et al., 2014a] with annotated instances either from still images or from video frames and test them on both domains. In this fashion, we can observe how the performance of a detector depends on the domain it is trained from.

#### 3.3.1 First dataset pair

For still images, we use VOC [Everingham et al., 2007], one of the most widely used datasets for object detection. For video frames we employ YouTube-Objects [Kalogiton et al., 2016; Prest et al., 2012a], which is one of the largest available video datasets with bounding-box annotations on multiple classes. It has 10 classes from VOC, which enables studying image-video domain differences.

**Still images (VOC).** Out of the 20 classes in VOC, we use the 10 which have moving objects, in order to have the same ones as in YouTube-Objects. Each object instance of these classes is annotated with a bounding-box in both training and test sets. Table 3.1 shows dataset statistics.

**Video frames (YTO).** The YouTube-Objects dataset [Kalogeiton et al., 2016; Prest et al., 2012a] contains videos collected from YouTube for 10 classes of moving objects. While it consists of 155 videos and over 720,152 frames, only 1,258 of them are annotated with one bounding-box around an object instance, i.e., three times fewer than in VOC. Instead, we would like to have a comparable number of annotations in both datasets. This would exclude differences in performance due to differences in the size of the training sets.

Therefore, we introduce the second version of YTO (v2), see Appendix A for more details. We first split the videos into disjoint training and test sets. Frames from the same video belong only to one set, avoiding any bias between training and test set. Then, for both sets, we uniformly sample a constant number of frames in each shot, so that the total number of YTO training samples is roughly equal to the number of VOC training samples. For the training set, we annotate one object instance per frame. For the test set, we annotate all instances. The total number of annotated samples is 6,973 (obtained from 6,087 frames). Figure 3.2 shows some annotated frames. The YTO v2.0-v.2.2 together with (a) the new annotations, (b) original videos with sound, (c) optical flow as produced by [Brox and Malik, 2011], and (d) superpixels as produced by [Achanta et al., 2012] are available online at <http://calvin.inf.ed.ac.uk/datasets/youtube-objects-dataset/>, and the YTO v2.3 (Dataset viewer), which provides the annotations in PASCAL VOC 2007 format for the same 6,973 bounding-box annotations from the YTO v2.2, is available at <https://github.com/vkalogeiton/yto-dataset>.

**Equalizing the number of samples per class.** For each class, we equalize the number of training samples *exactly*, by randomly sub-sampling the larger of the two training sets. The final number of equalized training samples is 3,907 in total over the 10 classes (see column ‘equalized’ in Table. 3.1). Only these equalized training sets will be used in the following. We refer to them to as trainVOC and trainYTO for still images and video frames, respectively.

When testing on VOC we use the complete VOC test set (Table 3.1; this includes also images without instances of our 10 classes). When testing on YTO, we use the

test set of 1,781 images with 2,667 objects instances in total (Table 3.1). We refer to them as testVOC and testYTO, respectively.

### 3.3.2 Second dataset pair

Both datasets in this pair come from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC [Russakovsky et al., 2015b]), i.e., from the object detection in images (IMG) and in videos (VID) tracks of the challenge. We consider the same 10 object classes as in the first pair of datasets.

**Still images (ILSVRC IMG).** The ILSVRC IMG dataset contains 60k training images (train60k) and 20k validation images, fully annotated with bounding-boxes on all instances of 200 object classes. We split the validation set into val1 and val2 as in Girshick et al. [2014a]. For training, we use train60k+val1, resulting in 13,335 bounding-boxes for our 10 classes (8,021 images). For testing we use val2, which comprises 5,310 bounding-boxes (3,362 images). We refer to it as ‘testDET’.

**Video frames (ILSVRC VID).** The ILSVRC VID dataset contains 3,862 training and 555 validation video snippets<sup>4</sup>, which we use for training and testing respectively. The snippets are manually annotated with bounding-boxes for 30 object classes. For our 10 classes, the training set has 2,198 snippets, totalling 292,199 bounding-boxes in 212,643 frames. The validation set, used as test set, has 332 snippets, with 134,432 bounding-boxes in 87,715 frames. We refer to it as ‘testVID’.

**Equalizing the number of samples per class.** To have the same number of training samples in each domain we apply the same equalization procedure as in the first pair of datasets. This results in 13,335 training samples and 3,362 test images per domain. We refer to the two training sets as trainIMG and trainVID.

### 3.3.3 Protocol

We want to train object detectors either from still images or from video frames and then test them on both domains. That means that each training set contains samples from

---

<sup>4</sup>Video snippets are small parts of a video each of them forming a unit. They differ from shots, as there is no systematic method for splitting the video into snippets.

one domain only. For each class, the positive training set contains annotated samples of this class, while the negative set contains images of all other classes.

We measure performance using the VOC protocol. A detection is correct if its intersection-over-union overlap with a ground-truth bounding-box is  $> 0.5$  [Everingham et al., 2007]. The performance for a class is Average Precision (AP) on the test set, and the overall performance is captured by the mean AP over all classes (mAP). Note here, that the video datasets we use contain only bounding box annotations, and therefore we cannot apply them to other tasks, such as semantic segmentation or part detection.

We experiment using two modern object detectors: Deformable Part Model (DPM [Felzenszwalb et al., 2010; Girshick et al., 2012b]) and Regions with Convolutional Neural Networks (R-CNN [Girshick et al., 2014a]).

DPM models an object class by a mixture of components, each composed of a root HOG template [Dalal and Triggs, 2005] and a collection of part templates arranged in a deformable configuration. This detector was the state-of-the-art reference for several years, until the arrival of CNN-based models. For more details see Section 2.2.1.

R-CNN [Girshick et al., 2014a] is one of the current leading object detectors. Candidate regions are obtained by selective search [Uijlings et al., 2013] and described with CNNs extracted with Caffe [Jia et al., 2014; Jia, 2013]. A linear SVM is then trained to separate positive and negative training regions (with hard negative mining to handle the large number of negative regions [Dalal and Triggs, 2005; Girshick et al., 2014a, 2012b]). For more details about R-CNN see Section 2.2.2. In this work, we use as features the 7<sup>th</sup> layer of the CNN model trained on the ILSVRC 2012 classification challenge [Krizhevsky et al., 2012], as provided by [Girshick et al., 2014b]. We do not fine-tune the CNN for object detection, so that the features are not biased to a particular dataset. This enables to measure domain shift factors more cleanly. Fine-tuning in a specific dataset would result in higher performance gaps and therefore, more misleading results.

| classname | Training            |      |           | Test                |      |
|-----------|---------------------|------|-----------|---------------------|------|
|           | # of object samples |      |           | # of object samples |      |
|           | VOC                 | YTO  | Equalized | VOC                 | YTO  |
| aeroplane | 306                 | 415  | 306       | 285                 | 180  |
| bird      | 486                 | 359  | 359       | 459                 | 162  |
| boat      | 290                 | 357  | 290       | 263                 | 233  |
| car       | 1250                | 915  | 915       | 1201                | 605  |
| cat       | 376                 | 326  | 326       | 358                 | 165  |
| cow       | 259                 | 321  | 259       | 244                 | 315  |
| dog       | 510                 | 454  | 454       | 489                 | 173  |
| horse     | 362                 | 427  | 362       | 348                 | 463  |
| motorbike | 339                 | 360  | 339       | 325                 | 213  |
| train     | 297                 | 372  | 297       | 282                 | 158  |
| total     | 4475                | 4306 | 3907      | 4254                | 2667 |

Table 3.1: Number of object samples in the training and test sets for image (VOC) and video (YTO) domains.

### 3.4 Analysis of domain shift factors and experimental results

For simplicity, we first focus on the first dataset pair, i.e., VOC and YTO (Sections 3.4.1–3.4.5). Results on the second pair (ILSVRC 2015) are reported in Section 3.4.6.

Here, we analyze the difference between VOC and YTO according to five factors: spatial location accuracy, appearance diversity, image quality, aspect distribution and others, such as object size and aspect-ratio and camera framing. We examine the first four factors by following the same procedure: *(1: measurement)* We introduce a metric to quantify the factor in each domain. *(2: equalization)* We present a way to make the training sets of the two domains more similar in terms of this metric. *(3: impact)* We compare the performance of object detectors trained from each domain before and after the equalization step. This enables to measure if, and by how much, cancelling out this factor has an impact on the performance.

For the last category of factors (Section 3.4.5) we do not observe any significant differences between VOC and YTO, and so we did not proceed to the equalization and impact steps.

As we apply the procedure above to each factor in sequence, we observe the evolution of the performance gap as the two domains are gradually equalized. As we have



two test sets (one per domain) we monitor the evolution of two performance gaps in parallel.

### 3.4.1 Spatial location accuracy

There are several methods to automatically segment objects from the background in video frames by exploiting spatio-temporal continuity [Brox and Malik, 2010; Lee et al., 2011; Papazoglou and Ferrari, 2013; Prest et al., 2012a]. We evaluate two methods: PRE, the method of Prest et al. [2012a], which extends the motion segmentation algorithm [Brox and Malik, 2010] to joint co-localization over all videos of an object class; and FVS, the Fast Video Segmentation method of Papazoglou and Ferrari [2013], which operates on individual videos. Both methods automatically generate bounding-boxes for all video frames. We sample as many bounding-boxes as there are in the trainVOC and trainYTO sets by following the approach in [Prest et al., 2012a].

In the first step we quantify the quality of each bounding-box, based on its objectness probability [Alexe et al., 2012] and the amount of contact with the image border (boxes with high contact typically contain background). In the second step we randomly sample bounding-boxes according to their quality (treating the quality values for all samples as a multinomial distribution). In this way, we obtain the PRE and FVS training sets.

In this section, we use the trainVOC set for still images. For video frames, we use the PRE and FVS training sets and we measure their accuracy with respect to the ground-truth annotations, see Section 3.4.1: Measurement. We also use the trainYTO set, in order to improve video training data to match the perfect spatial support of still images (Section 3.4.1: Equalization). Finally, we train object detectors from each training set (trainVOC and trainYTO) and test them on testVOC and testYTO. In this way, we can quantify the impact of the different levels of spatial location accuracy on performance (Section 3.4.1: Impact).

**Measurement.** We measure the accuracy of bounding-boxes by CorLoc: the percentage of bounding-boxes that satisfy the PASCAL VOC criterion [Everingham et al., 2010] ( $\text{IoU} > 50\%$ ). Bounding-boxes computed by the PRE method have 24.0% CorLoc, while FVS brings 54.3% CorLoc. This shows that FVS can automatically produce good bounding-boxes in about half the frames, which is considerably better than the ones from PRE (Figure 3.3). However, this is worse than having all frames correctly



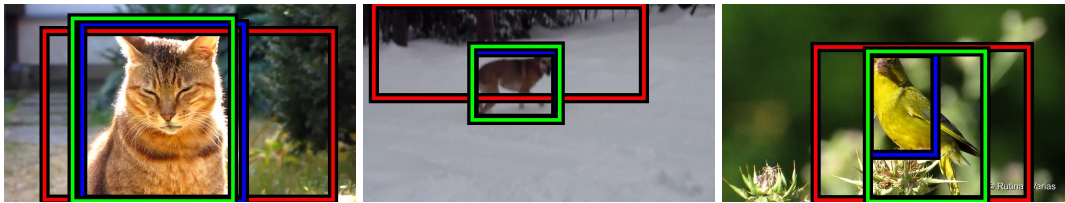


Figure 3.3: Example bounding-boxes produced by PRE [Prest et al., 2012a] (red), FVS [Papazoglou and Ferrari, 2013] (blue), and ground-truth annotations (green).

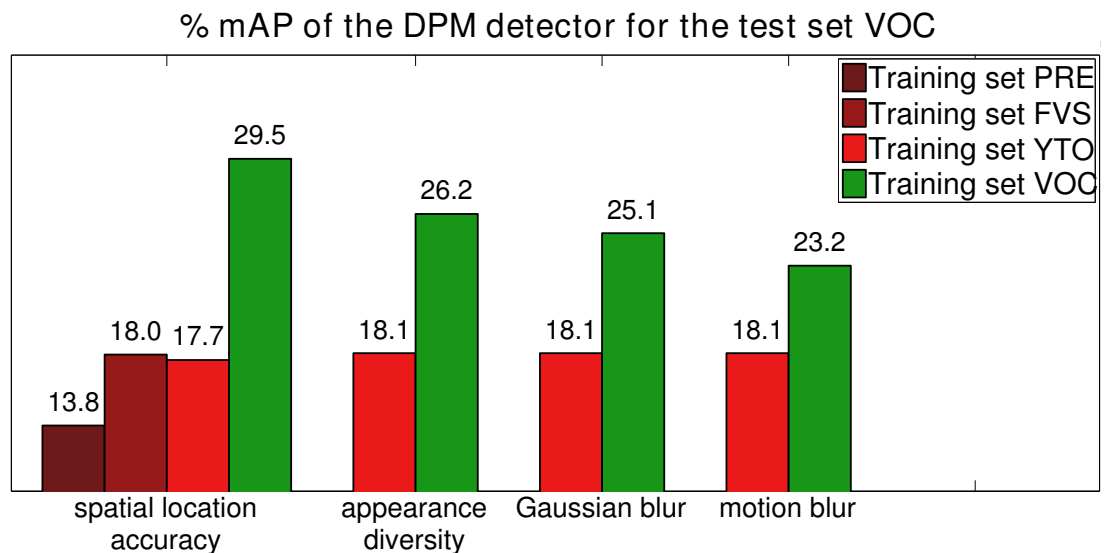
annotated, as it is the case with manual ground-truth in trainYTO.

**Equalization.** The equalization step enhances the quality of the bounding-boxes in video frames, gradually moving from the worst to perfect annotations. We match the perfect location accuracy of still images (trainVOC) by using ground-truth bounding-boxes for the video frames (trainYTO).

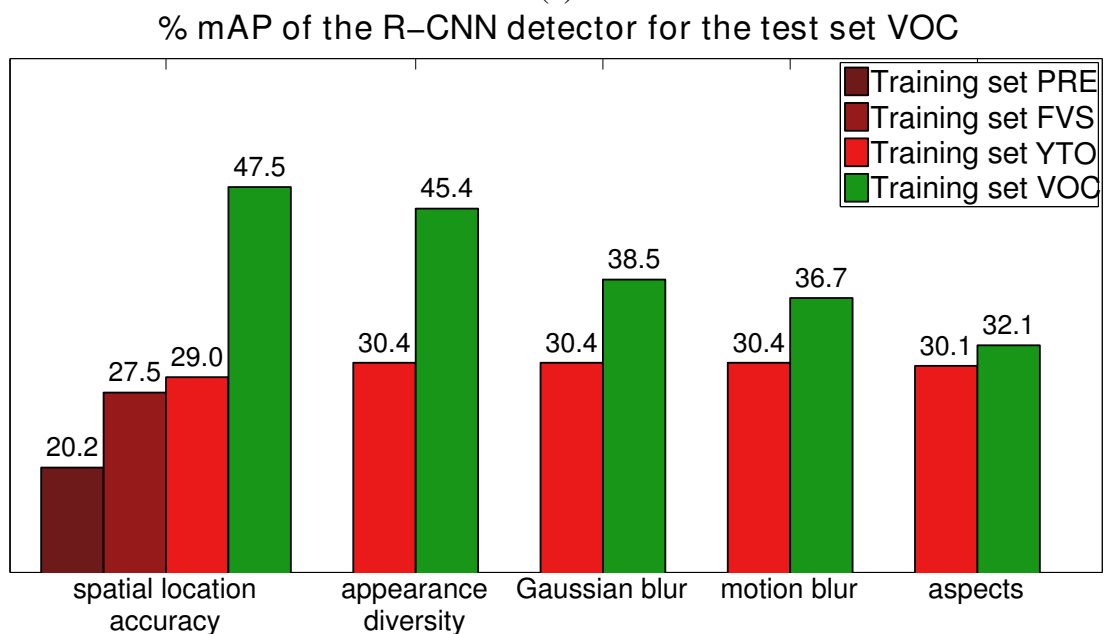
**Impact.** For video frames we train object detectors for each of the three levels of spatial support: starting with poor automatic annotations (PRE), then moving to better ones (FVS), and finally using ground-truth bounding-boxes (trainYTO). For still images we train detectors with ground-truth bounding-boxes of (trainVOC). We test on the testVOC and testYTO sets. Figures 3.4–3.5 report the mAP performance averaged over all classes for both detectors (DPM, R-CNN) for testVOC and testYTO, respectively. For per-class results, refer to Appendix B.

When testing on still images (testVOC), the mAP of training from video continuously improves when using more and more accurate spatial support (Figure 3.4). However, training on trainVOC leads to a considerably superior performance to training on videos, even when training with the perfect ground-truth annotations of trainYTO.

These results show that the imperfect spatial location accuracy of training samples produced by automatic video segmentation methods can only explain part of the gap. This is surprising, as we expected that using perfect annotations would close the gap much more. Quantitatively, for DPM the gap goes from 15.7% to 11.8% when going from training on the weakest automatic segmentation (PRE) to ground-truth bounding-boxes (trainYTO). The result is analogous for R-CNN, with the gap going from 27.3% when using PRE, to 18.5% when using trainYTO. These results imply that we cannot get detectors learned from video to perform very well on still images even with great future progress on video segmentation, and in fact not even by manually annotating



(a)

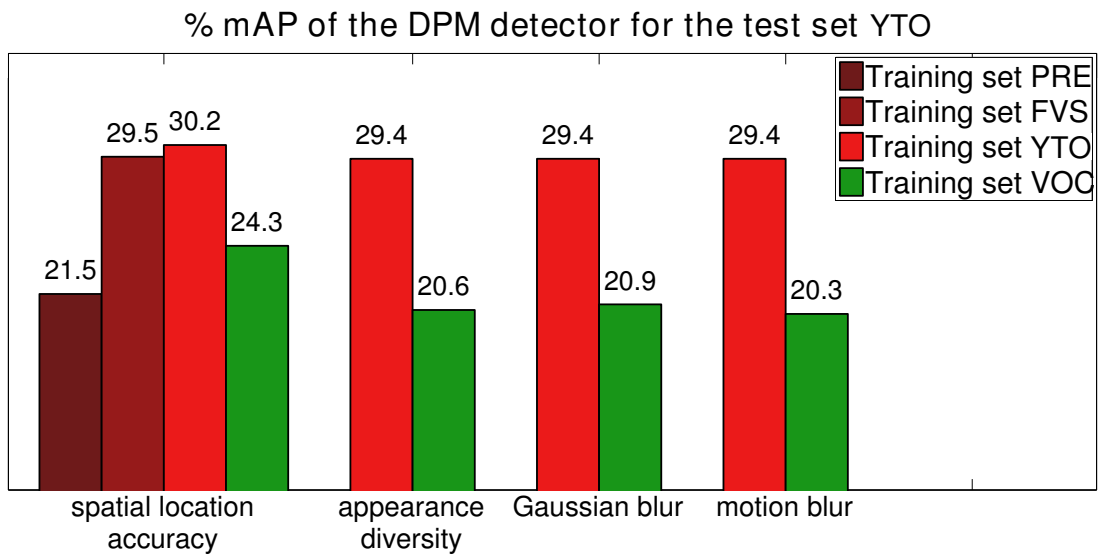


(b)

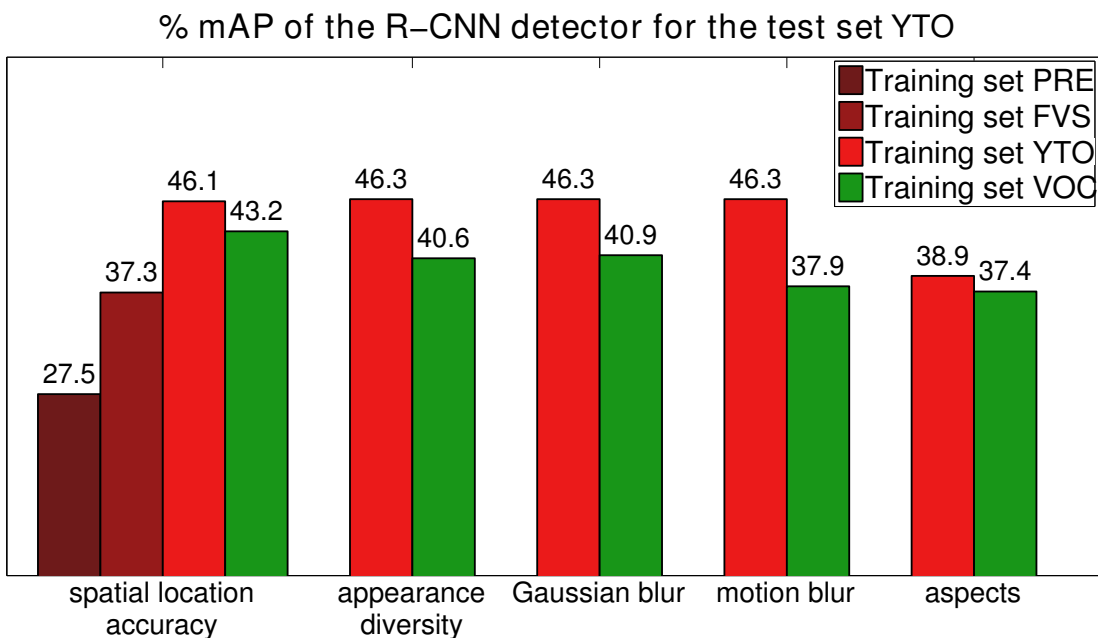
Figure 3.4: *mAP* results: impact of the domain shift factors when training on VOC and YTO (a) the DPM detector and (b) the R-CNN detector for the test VOC.

frames. Moreover, this also suggests there are other significant causes that produce the leftover gap.

Testing on videos (testYTO) reveals a similar trend: more accurate spatial support on video frames leads to better performance (Figure 3.5). Interestingly, training from videos here performs better than training from still images (when both training sets are ground-truth annotated). This shows that we are confronted with a real domain



(a)



(b)

Figure 3.5: *mAP results: impact of the domain shift factors when training on VOC and YTO (a) the DPM detector and (b) the R-CNN detector for the test YTO.*

adaptation problem, where it is always better to train on the test domain. Again results hold for both detectors, but the ‘reverse gap’ left after equalizing the spatial location accuracy is smaller than on testVOC: 5.9% mAP for DPM and 3.6% for R-CNN. Note also, that for R-CNN the gap is smaller 3.6% compared to the 5.9% for DPM. This is probably due to the fact that in R-CNN the network is pre-trained on images, thus rendering the performance of videos lower than if it was pre-trained on videos.



Figure 3.6: (top row) YTO dataset: Frames in the same shot that contain near identical samples of an object. (bottom row) VOC dataset: Example of near identical samples in the same image.

### 3.4.2 Appearance diversity

Video is intrinsically different from still images in that frames are temporally correlated. Frames that are close in time often contain near identical samples of the same object (top row of Figure 3.6). In still images such repetition happens rarely and typically samples that look very similar co-occur in the same image (bottom row of Figure 3.6). We first measure the appearance diversity of training sets (Section 3.4.2: Measurement). Then we modify them to equalize their appearance diversity (Section 3.4.2: Equalization). Finally, we observe the impact of this equalization on the performances of object detectors (Section 3.4.2: Impact). In the spirit of our progressive equalization mission, here we use the trainYTO and trainVOC sets, which have ground-truth annotations. In this way, we focus on differences due to appearance diversity alone and not due to spatial support.

**Measurement.** To measure appearance diversity within a training set, we manually group near-identical samples, i.e., samples of objects in very similar viewing conditions (e.g., viewpoint and degrees of occlusion, Figure 3.6). This results in a set of groups, each containing near-identical samples (Figure 3.7). We quantify appearance

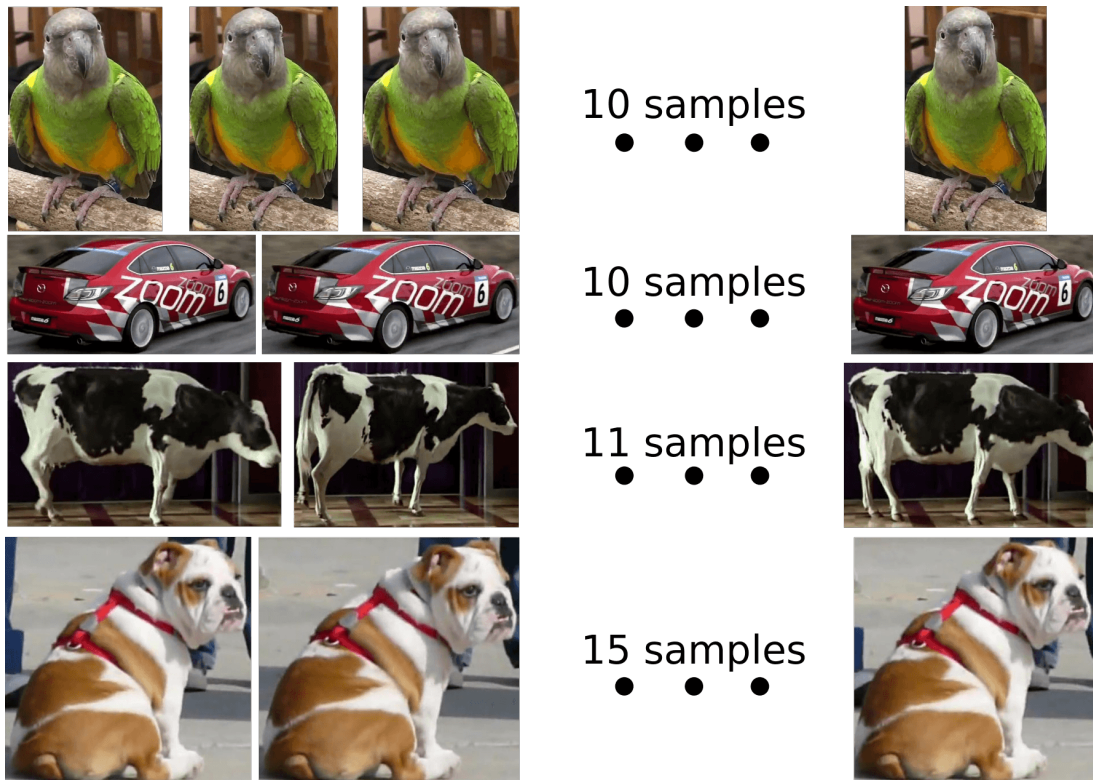


Figure 3.7: Four example groups of near-identical samples in trainYTO. We display a subset of the frames for each group.

diversity by the number of groups, i.e., the number of *unique samples* in the training set.

As shown in Table 3.2, trainYTO has only half the number of unique samples than trainVOC, despite them having exactly the same total number of samples (Table 3.1). This shows that half of the video samples (51%) are repeated, while almost all (97%) still image samples are unique. This reveals a considerable difference in appearance diversity between the two domains.

**Equalization.** We equalize appearance diversity by resampling each training set so that: (1) it contains only unique samples; and (2) the size of the training sets is the same in the two domains. We achieve the first goal by randomly picking one sample per group, and the second by randomly subsampling the larger of the two training sets (i.e. VOC). This procedure is applied for each class separately. This leads to the new training sets ‘trainVOC Unique Samples’ and ‘trainYTO Unique Samples’, each containing 2,201 unique samples (Table 3.2, column ‘Equalized Unique Samples’).



**Impact.** We train object detectors from the equalized unique sample sets only. Figures 3.4–3.5 report results for both detection models and test sets.

Figure 3.4 reports the performance when testing on VOC. We observe that the mAP of training from still images decreases significantly when going from using all training samples (trainVOC) to unique samples (trainVOC Unique Samples), as about half of the unique training samples are removed. Instead, the mAP of training from videos remains almost constant, as only duplicate samples are removed.

Figure 3.5 reports the performance when testing on YTO. We observe that testing on YTO produces similar effects compared to testing on VOC. In particular, the unique sample equalization procedure leaving the performance of training from YTO almost unchanged, but significantly reducing that of training from VOC. These results reveal that indeed near identical samples do not bring any extra information, and only artificially inflate the apparent size of a training set. Hence, these findings suggest that one should pool training samples out of a large set of diverse videos, sampling very few frames from each shot.

Equalizing appearance diversity reduces the performance gap when testing on VOC down to 8.1% mAP for DPM and 15.0% mAP for R-CNN. Notably, this bridges the gap for both detectors by about the same amount (3.5% – 3.7%). When testing on YTO the equalization has the opposite effect and increases the gap by about 3% to 8.8% mAP for DPM and 5.7% for R-CNN. This is expected, as the process handicaps still images (trainVOC) down to the level of diversity of videos (trainYTO), without harming videos (trainYTO).

### 3.4.3 Image quality

We examine the image quality factor while working on the unique samples training sets, which have the same size, accuracy of spatial support, and level of appearance diversity. In this way, all those factors will not cause any performance difference.

**Measurement.** We measure the image quality of a training sample by its gradient energy, as in [Prest et al., 2012a]. This computes the sum of the gradient magnitudes in the HOG cells of an object bounding-box, normalized by its size (computed using the implementation of Felzenszwalb et al. [2010]). The gradient energy averaged over all classes is 4.4 for unique samples of images (trainVOC Unique Samples) and 3.2 for the unique samples of videos (trainYTO Unique Samples). This difference of gradient

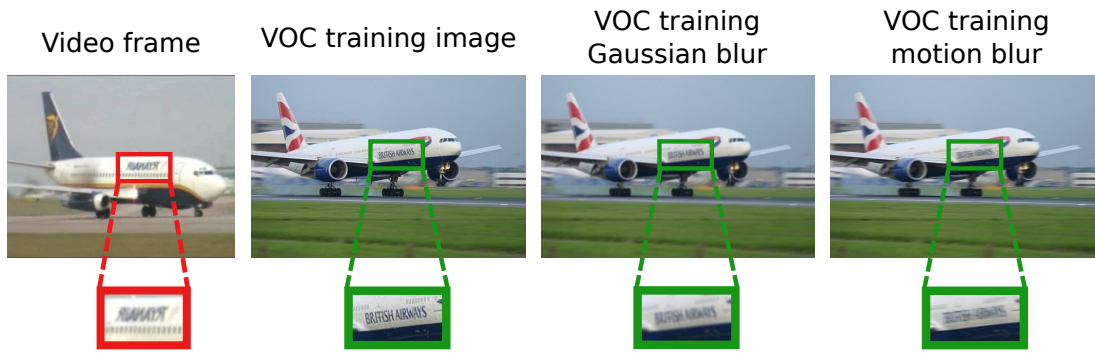


Figure 3.8: Video frame, VOC training image, Gaussian and motion blurred VOC training images.

energy between images and videos is because video frames suffer from compression artefacts, motion blur, and low color contrast.

**Equalization.** We equalize the gradient energy by blurring the VOC samples, so as to match the energy of the YTO samples. We consider two different ways to blur a sample: Gaussian blur and motion blur. For Gaussian blur we apply an isotropic Gaussian filter with standard deviation  $\sigma$ . For Motion blur we apply a box filter of length  $K$  along the horizontal direction, as most camera motion in YouTube videos is horizontal. We also experimented with vertical direction without much difference in the performance. The motion blurred value  $g(m, n)$  of a pixel  $(m, n)$  is given by:

$$g(m, n) = \frac{1}{K} \sum_{i=0}^{K-1} f(m-i, n)$$

We set the parameters of the blur filter ( $\sigma$  and  $K$ ) separately for each class, so that the average gradient energy of the blurred image samples (VOC) equals that of the video samples (YTO). We find the exact parameter values using a bisection search algorithm (as an indication, the average values are  $\sigma = 1.35$  and  $K = 8.4$ ). This procedure leads to the new training sets ‘trainVOC Gaussian Blurred Unique Samples’ and ‘trainVOC Motion Blurred Unique Samples’. For uniformity, we also apply the same blur filters to the negative training sets. Figure 3.8 shows the effect of the blur filters on a VOC training image.

**Impact.** We train object detectors from each of the two trainVOC blurred Unique Samples sets. Figures 3.4–3.5 report results for both detection models for testVOC

| classname | Number of groups |      | Ratio |      | Equalized      |
|-----------|------------------|------|-------|------|----------------|
|           | YTO              | VOC  | YTO   | VOC  | Unique Samples |
| aeroplane | 244              | 268  | 0.59  | 0.88 | 244            |
| bird      | 123              | 452  | 0.34  | 0.93 | 123            |
| boat      | 138              | 275  | 0.39  | 0.95 | 138            |
| car       | 310              | 1221 | 0.34  | 0.98 | 310            |
| cat       | 249              | 376  | 0.76  | 1.00 | 249            |
| cow       | 90               | 252  | 0.28  | 0.97 | 90             |
| dog       | 295              | 507  | 0.65  | 0.99 | 295            |
| horse     | 286              | 358  | 0.67  | 0.99 | 286            |
| motorbike | 243              | 337  | 0.68  | 0.99 | 243            |
| train     | 223              | 294  | 0.60  | 0.99 | 223            |
| avg       | 220              | 434  | 0.51  | 0.97 | 220            |

Table 3.2: Appearance diversity equalization. Statistics of the groups: number of groups, ratio: number of groups / number of ground-truth samples and number of equalized unique samples.

and testYTO, respectively. Note how results do not change when training from YTO, as this equalization process does not affect video training data.

Figure 3.4 reports the mAP when testing on image. We observe that performance drops considerably when using blurred training samples, especially for R-CNN. On both detection models, the effect is more pronounced for motion blur than for Gaussian blur. This is likely because motion blur rarely happens naturally in still images, and so it is almost entirely absent in testVOC, making the equalization process distort the training set further away from the test set statistics. This also reveals that motion blur is a more important domain difference between VOC and YTO than Gaussian blur.

Figure 3.5 reports the performance when testing on videos. The performance when testing on YTO shows an interesting phenomenon: using blurred training samples has a much smaller effect. This is normal as testYTO is already naturally blurred, and therefore blurring the training set does not lose much relevant information.

Equalizing image quality with Gaussian blur reduces the performance gap when testing on VOC down to 7.0% mAP for DPM and 8.1% for R-CNN. Motion blur makes the gap even smaller: 5.1% for DPM and 6.3% for R-CNN. The amount of gap bridged for R-CNN is remarkably large (8.7% mAP). When testing on YTO, Gaussian



blur leaves the gap essentially unchanged for both detectors, while motion blur widens the gap for R-CNN by a small amount of 2.7%, reaching 8.4% mAP. Given that motion blur better represents the relevant image quality difference between the two domains, in the following we work only with motion blurred training sets.

### 3.4.4 Aspect distribution

As the last factor, we consider the distribution over aspects, i.e., the type of object samples in the training sets. Differences can be due to biases in the distribution of viewpoints, subclasses, articulation and occlusion patterns. As the space of possible samples for an object class is very large, any given dataset invariably samples it in a limited way, with its own specific bias [Torralla and Efros, 2011]. Figure 3.9 illustrates this point by showing all training samples of the class ‘horse’ from both domains. The distributions differ considerably and overlap only partially. Horses jumping over hurdles appear in trainVOC but not in trainYTO, while the latter has more horses running free in the countryside (more examples in Figures 3.11-3.12). We work here with the most equalized training sets, i.e. trainVOC Motion Blurred Unique Samples and trainYTO Unique Samples. These have the same size (number of samples in each set), accuracy of spatial support, level of appearance diversity, and image quality.

**Measurement.** We refer to the two training sets as  $A = (x_1^A, \dots, x_n^A)$  for VOC, and  $B = (x_1^B, \dots, x_n^B)$  for YTO. Measuring the difference in distributions of a source and a target domain is not a new task. Hoffman et al. [2014a] learn a similarity function by performing feature transformations. Duan et al. [2012a] measure the distribution mismatch based on the distance between the mean values of the two domains, referred to as Maximum Mean Discrepancy [Borgwardt et al., 2006]. Here, we measure the difference in the aspect distributions of the two training sets with the symmetrized Kullback-Leibler (KL) divergence

$$d_{\text{KL}}(A, B) = D_{\text{KL}}(A, B) + D_{\text{KL}}(B, A) \quad , \text{ where} \quad (3.1)$$

$$D_{\text{KL}}(A, B) = \sum_{i=1}^n \hat{f}_A(x_i^A) \ln \frac{\hat{f}_A(x_i^A)}{\hat{f}_B(x_i^B)} \quad , \text{ and} \quad (3.2)$$

$$\hat{f}_s(x^s) = \frac{1}{n} \sum_{i=1}^n K(x^s - x_i^s; h) \quad , \quad s \in \{A, B\} \quad (3.3)$$

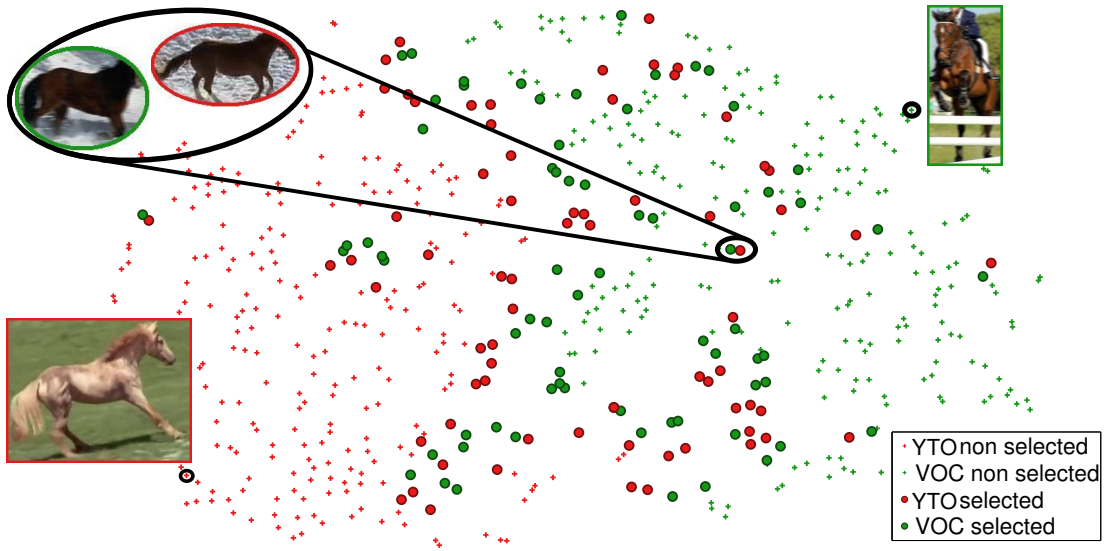


Figure 3.9: 2D visualization of the trainYTO Unique Samples (red) and trainVOC Motion Blurred Unique Samples (green) for the ‘horse’ class in R-CNN feature space. Circles indicate the samples selected by our equalization technique of Section 3.4.4: Equalization.

is a kernel density estimator fit to sample set  $s$ , and  $K(\cdot; h)$  is the isotropic Gaussian kernel with standard deviation  $h$  (automatically set based on the standard deviation of the sample set [kde, 2003]). For ease of visualization and computational efficiency, we reduce the dimensionality of the CNN features to 2D, using the algorithm of [Van der Maaten and Hinton, 2008], as done by [Donahue et al., 2014; Jia et al., 2014]. Note that, we also reduced the dimensionality to 64D and 128D using PCA and observed the same trend in the detection performance. We did not use the original 4096D space, as quantifying discrepancies in the distributions in such a high-dimensional space it rendered difficult.

The symmetric KL divergence between the two training sets, averaged over all classes, is 5.25. This shows that the difference in aspect distribution is quite big, given that the KL divergence averaged over all classes between trainVOC and testVOC is 1.24.

The symmetric KL divergence captures the discrepancy between any two distributions. Therefore, we also examined the evolution of the KL divergence before and after the previous equalization steps of Sections 3.4.2-3.4.3: we observed that the KL divergence drops after every equalization step, validating the effectiveness of our equalization process.

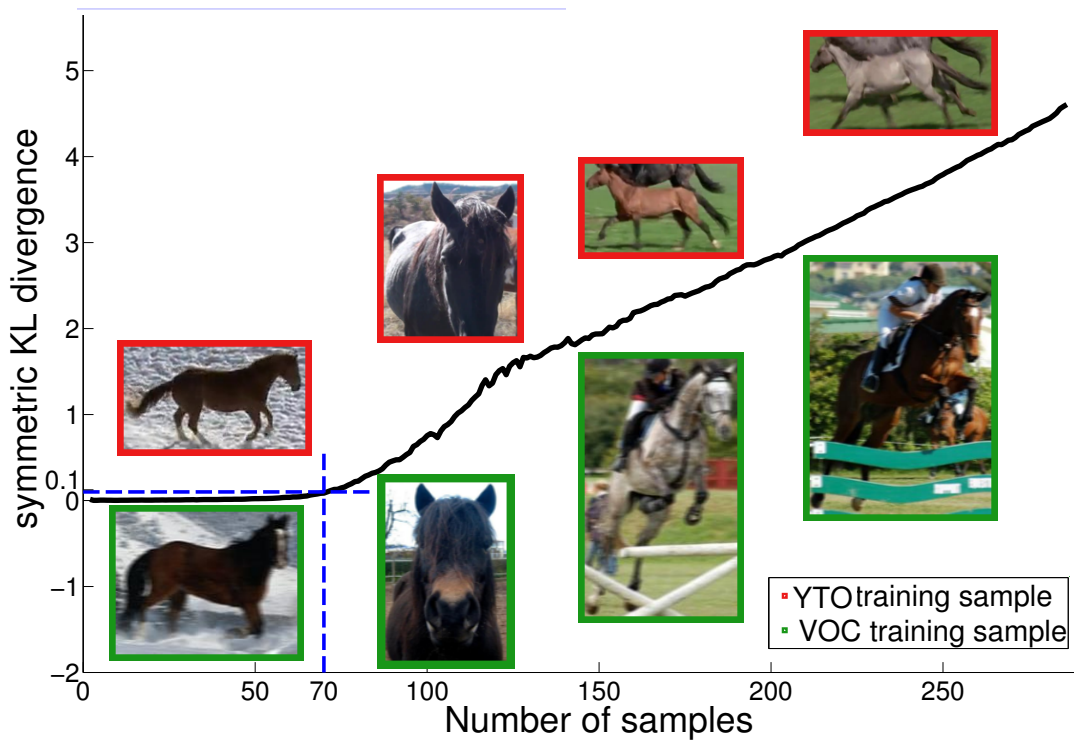


Figure 3.10: Evolution of  $d_{KL}$  between the image and video training sets as for each training set our algorithm adds more and more sample pairs. The two distributions are very similar at the beginning and start to diverge later. The plot corresponds to the object horse, where at the  $\varepsilon = 0.1$  threshold 70 samples from each set are selected. This number is driven by  $\varepsilon$  and changes from class to class.

**Equalization.** We equalize the aspect distributions by subsampling the two training sets such that the subsets have a similar aspect distribution, i.e., a small  $d_{KL}$ . More precisely, we want to find the largest subsets  $\tilde{A} \subset A$  and  $\tilde{B} \subset B$  which have a small enough  $d_{KL}$  to be considered equally distributed:  $d_{KL}(\tilde{A}, \tilde{B}) < \varepsilon$ . We approximate this optimization by a greedy forward selection algorithm that starts from  $\tilde{A} = \tilde{B} = \emptyset$ , and iteratively adds the pairs of samples with the smallest Euclidean distance. We stop growing the subsets when  $d_{KL}$  exceeds  $\varepsilon$ .

Figure 3.10 illustrates the evolution of  $d_{KL}(\tilde{A}, \tilde{B})$  during this process for the class ‘horse’. We use a small  $\varepsilon = 0.1$  in all experiments. Note the need for this parameter, otherwise  $d_{KL}(\tilde{A}, \tilde{B}) = 0$  is achieved by picking 0 samples from each set. For the horse class, this process selects 70 samples from each set, which is just after the horizontal portion of the curve in Figure 3.10, when the distributions start to differ significantly, see samples along the curve. Figure 3.9 depicts a selected pair of samples, which lies

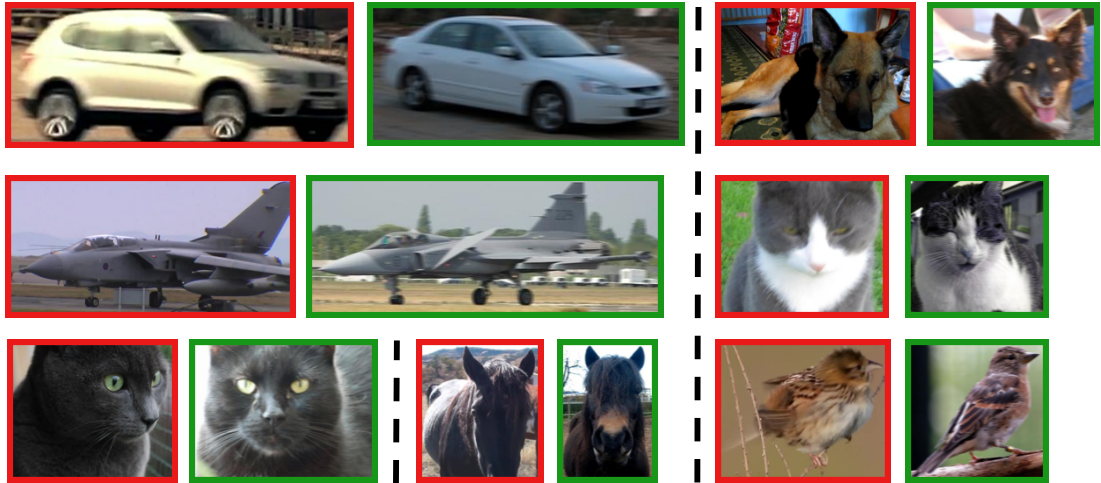


Figure 3.11: Aspects common to both VOC (green) and YTO (red). Both datasets contain samples with same aspects, such as cat faces, dog sitting, or horse running.

in the region where the distributions overlap. This pair shows samples with similar aspects, whereas distant samples typically show very different aspects.

This procedure constructs the new training sets ‘trainVOC Motion Blurred Unique Samples and Aspects’ and ‘trainYTO Unique Samples and Aspects’. These sets contain 551 samples each (about 1/4 of all samples left after equalizing the previous domain shift factors).

**Impact.** We train object detectors from the aspect-distribution equalized sample sets. Results for R-CNN are reported in Figure 3.4 (b) for testVOC and in Figure 3.5 (b) for testYTO. We do not perform this equalization process for DPM, as it does not work on a simple feature space where we can easily measure distances, due to its movable parts.

Figure 3.4 (b) shows that when testing on VOC, the performance of training from YTO is barely affected by the equalization process, despite having  $4\times$  fewer training samples. Instead, the mAP of training from VOC drops considerably. This can be explained by the fact that the equalization process returns the intersection of the distributions of the two training sets. The video training set only loses samples with aspects not occurring in the trainVOC distribution, and hence unlikely to be in the test set. Instead, the VOC training set is losing many aspects that do occur in the test set.

Testing on YTO (Figure 3.5 (b)) corroborates this interpretation by displaying the inverse behavior: the performance of training on VOC remains essentially unchanged, whereas that of training on YTO worsens substantially.



Figure 3.12: *Different aspects between VOC (green) and YTO (red). Top row: aspects occurring only in VOC. Chicken and birds flying are common in VOC, but do not appear in YTO. Bottom row: aspects occurring only in YTO. YouTube users often film their own pets doing funny things, such as jumping, rolling and going on a skateboard.*

Equalizing the aspect distributions when testing on VOC, brings the performance down to just 2.0% mAP, closing the gap by 4.3%. When testing on YTO, the equalization has an even greater effect: it bridges the gap by 6.9% mAP, reducing the performance gap to just 1.5%.

The results show that aspects play an important role. Performance depends considerably on the training set containing aspects appearing in the test set, and this matters more than the size of the training set. The results also show that the aspect distributions in trainVOC and trainYTO are quite different, a dataset bias phenomenon analog to that observed by [Torralba and Efros \[2011\]](#) when studying the differences between still image datasets. Our findings provide a guideline for practitioners trying to enrich still image training sets with video data (or vice versa): it is more important to carefully consider *which* data samples to add, rather than simply trying to add a large number of them.

### 3.4.5 Other factors

In addition to the four domain shift factors we studied in Sections 3.4.1 to 3.4.2, we also considered other factors, which we summarize here. However, when measuring these other factors, we did not observe any significant differences between VOC and YTO, and so we did not proceed to the equalization and impact steps.



**Object size and aspect-ratio.** The average size of ground-truth bounding-boxes in trainVOC, relative to the size of the image, is 0.26. For trainYTO it is nearly the same (0.25). Similarly, the average aspect-ratio (width-over-height) is 1.48 for trainVOC vs 1.53 for trainYTO. Note that not only the average but also the distribution over classes of the measurements (object size and aspect-ratio) is the same between the two datasets.

**Camera framing.** We look for differences in the way objects are framed by the camera. Potentially, YTO might have more objects coming in and out of the frame. Each VOC instance is annotated by a tag marking it as either normal, truncated (partially out of the image frame), or difficult (very small, very dark, or heavily occluded) [Everingham et al., 2007]. In order to measure camera framing for trainYTO, we annotated all its instances with the same tags. Both trainVOC and trainYTO have about the same proportion of truncated instances (35.8% in trainVOC, 33.2% in trainYTO). We exclude instances marked as difficult from trainVOC, as they are not taken into account in the PASCAL VOC 2007 either. Only 0.3% of the trainYTO set are difficult instances, again leading to about the same percentage. All other instances are normal (i.e. about 65% in both trainVOC and trainYTO).

### 3.4.6 Experiments on the ILSVRC 2015 dataset pair

To verify that our findings are general we repeat here our analysis on the second dataset pair, as described in Section 3.3.2. Note that we consider the same 10 object classes as in the first pair of datasets. Following the protocol of Section 3.3.3, we train an R-CNN detector either from still images or from video frames, then test it on both domains, and finally measure performance by mAP on the test set.

**Domain shift factors.** We analyze 3 out of the 4 domain shift factors from Section 3.4. We do not examine the spatial location accuracy factor, since we start from perfect spatial support (ground-truth bounding-boxes). For the appearance diversity factor, 96.6% of the samples in trainIMG are unique, whereas only 61.6% trainVID samples are unique, analog to what observed on the VOC-YTO dataset pair. We apply the equalization procedure of Section 3.4.2, obtaining two new training sets, each containing 7,902 unique samples. For image quality, the gradient energy averaged over all classes is 4.4 for the unique samples in ILSVRC IMG (identical to VOC) and 3.0 for those in ILSVRC VID (i.e. blurrier than YTO). By applying the Gaussian blur

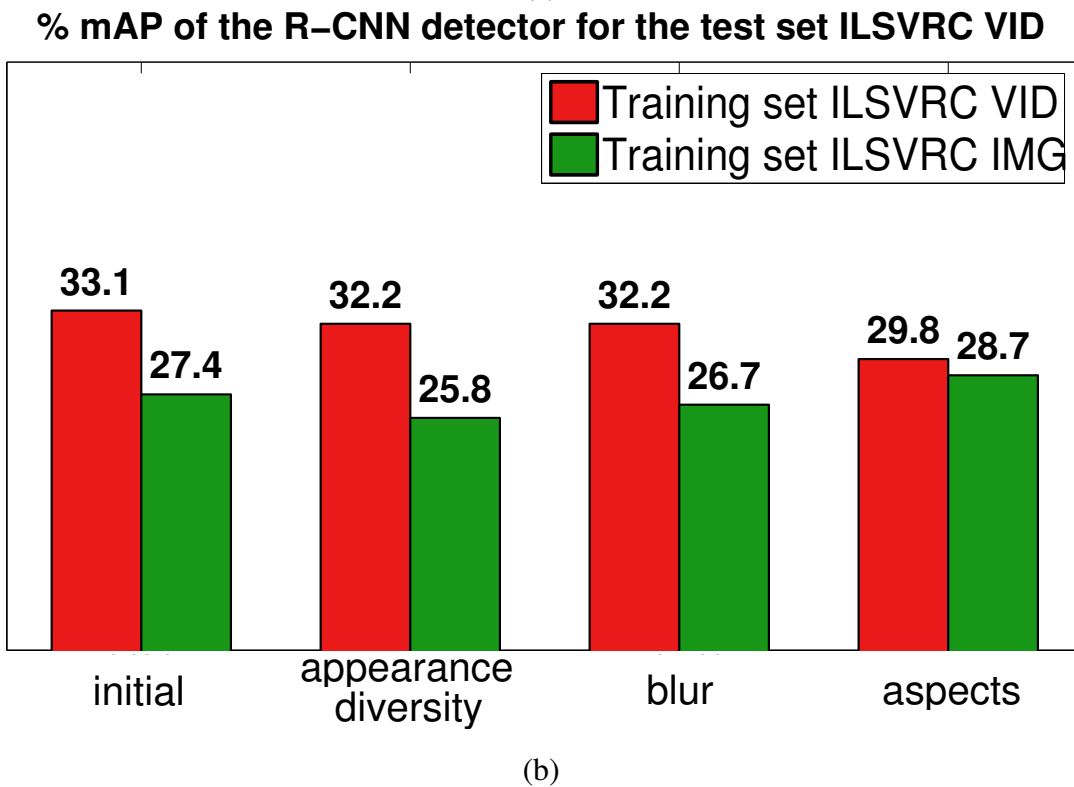
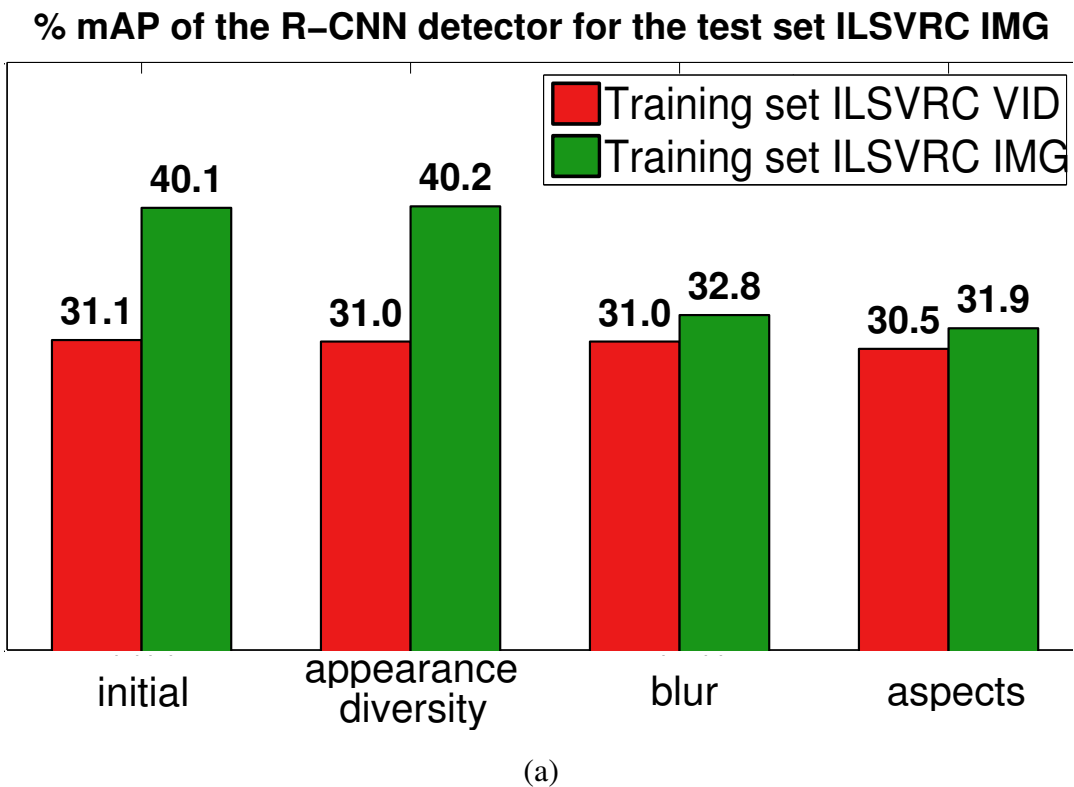


Figure 3.13: *mAP* results: impact of the domain shift factors when training on ILSVRC IMG and VID the R-CNN detector and testing on (a) ILSVRC IMG, and (b) ILSVRC VID.

filter of Section 3.4.3 on the image samples, we equalize their blur level to match the VID samples. For aspect distribution, the KL divergence between the two training sets, averaged over all classes, is 7.52. We apply the aspect distribution equalization procedure of Section 3.4.4, resulting in the two final training sets, each containing 3,446 samples.

**Impact of the factors.** Figure 3.13 shows the evolution of the performance of the R-CNN detector on the test sets after canceling out each domain shift factor in turn. For per-class results, refer to Appendix B. Generally, we observe the same trend as in the VOC-YTO pair, i.e., the gap is initially rather substantial, and it is gradually reduced by our equalization steps. The final gap after all steps is below 1.5% mAP on both test sets.

When looking closer, some differences to the VOC-YTO results appear. When testing on images, the appearance diversity factor leaves the gap unchanged. This is due to the larger number of training samples in ILSVRC IMG, compared to VOC ( $4\times$  more). Even after removing about 40% of the unique training samples from ILSVRC IMG in order to match the number of unique samples in ILSVRC VID, there are still enough samples left to train good detectors. Interestingly, when testing on images, the image quality factor closes the gap by a large margin. This is due to ILSVRC VID being blurrier than YTO, so the image quality equalization applies a stronger blur to ILSVRC IMG than to VOC. The aspect distribution factor bridges the performance gaps for both domains, in line with what observed on VOC-YTO. This confirms the important impact that the aspects contained in a training set have on performance at test time.

## 3.5 Conclusions

We analyzed several domain shift factors between still images and video frames for object detection. This is the first study that addresses with a systematic experimental protocol such an important task. We thoroughly explored four domain shift factors and their impact on the performance of two modern object detectors [Felzenszwalb et al., 2010; Girshick et al., 2014a]. We showed that by progressively cancelling out these factors we gradually closed the performance gap between training on the test domain and training on the other domain.

Given that data is becoming abundant, it is important to decide *which* data to anno-



tate to create better object detectors. Our experiments lead to several useful findings, especially relevant when trying to train detectors from video to perform well on image test sets: (1) training from videos with ground-truth bounding-box annotation still produces a worse detector than when training from still images. Hence, future research on video segmentation cannot solve the problem *on its own*; (2) blur has a strong impact on the performance gap; hence, deblurring algorithms might be an avenue for removing this factor; (3) the appearance diversity and aspect distribution of a training set is much more important than the number of training samples it contains. For good performance one should collect a broad range of videos showing all aspects expected to appear in the test set. In Appendix B we report per-class results for all the experiments of this chapter. Moreover, in Appendix C we present additional experiments when training object detectors from both still images and video frames.

# Chapter 4

## Object and action detection in videos

### Contents

---

|            |  |            |
|------------|--|------------|
| <b>4.1</b> | <b>Overview . . . . .</b>  | <b>87</b>  |
| <b>4.2</b> | <b>Related Work . . . . .</b>  | <b>89</b>  |
| <b>4.3</b> | <b>End-to-end multitask network architecture for joint learning of objects and actions . . . . .</b> | <b>91</b>  |
| 4.3.1      | End-to-end network architecture . . . . .  | 91         |
| 4.3.2      | Joint learning of objects and actions . . . . .  | 93         |
| <b>4.4</b> | <b>Experimental results for multitask learning . . . . .</b>   | <b>95</b>  |
| 4.4.1      | Joint detection of objects and actions in videos . . . . .   | 95         |
| 4.4.2      | Zero-shot learning of actions . . . . .  | 99         |
| 4.4.3      | Object-action segmentation . . . . .   | 100        |
| 4.4.4      | Relationship detection of objects and actions . . . . .  | 103        |
| <b>4.5</b> | <b>Exploring action adaptation techniques for action localization . . . . .</b>                      | <b>106</b> |
| 4.5.1      | Action adaptation . . . . .  | 107        |
| 4.5.2      | Experimental results on action adaptation . . . . .  | 110        |
| 4.5.3      | Action adversarial adaptation . . . . .  | 113        |
| 4.5.4      | Experimental results on action adversarial adaptation . . . . .                                      | 115        |
| <b>4.6</b> | <b>Conclusions . . . . .</b>   | <b>117</b> |

---

The efforts of the computer vision community have certainly led to impressive results on object detection [Ren et al., 2015a] and action recognition [Weinzaepfel et al., 2015]. For both tasks, the community has moved from small datasets [Rodriguez et al., 2008] to large ones with thousands of videos and hundreds of classes [Rusakovsky et al., 2015b; Heilbron et al., 2015], from controlled environments [Schüldt et al., 2004] to videos in-the-wild [Karpathy et al., 2014]. Given the impressive success of CNNs for object detection [Ren et al., 2015a; Liu et al., 2016a] (Section 2.2), action localization has benefited as well from this improvement. In particular, Faster R-CNN [Ren et al., 2015a] has been enhanced for videos by using a two-stream variant [Gkioxari and Malik, 2015; Peng and Schmid, 2016; Weinzaepfel et al., 2015] (Section 2.4), in which both appearance and motion are used as inputs. Modern approaches first use such a detector to localize human actions in individual frames, and then either link them or track them over time to create spatio-temporal detections [Gkioxari and Malik, 2015; Peng and Schmid, 2016; Weinzaepfel et al., 2015]. These methods focus exclusively on human action recognition.

While objects or actions alone are building blocks of video understanding, the relationship between objects and actions can yield a more complete interpretation. Therefore, in this chapter, we propose to jointly detect object-action instances in uncontrolled videos, e.g., *cat eating*, *dog running*, or *car rolling* (Figure 4.1).

To this end, we build an end-to-end two stream network architecture for detecting:

1. Objects and actions jointly: we cast this joint problem by leveraging a multitask objective.
2. Objects alone: we show that our multitask architecture improves on learning objects alone.
3. Actions alone: we examine various two-stream adaptive fusion techniques and observe their impact on action localization performance.

This work is published in [Kalogeiton et al., 2017b] and the evaluation code is available at <https://github.com/vkalogeiton/joint-object-action-learning>.

**Outline.** This chapter is organized as follows. We first present an overview of our method (Section 4.1), and then, we review related works in Section 4.2. Next, Section 4.3 describes our end-to-end multitask architecture for training an object-action detector. We present extensive experimental results on object detection alone and on

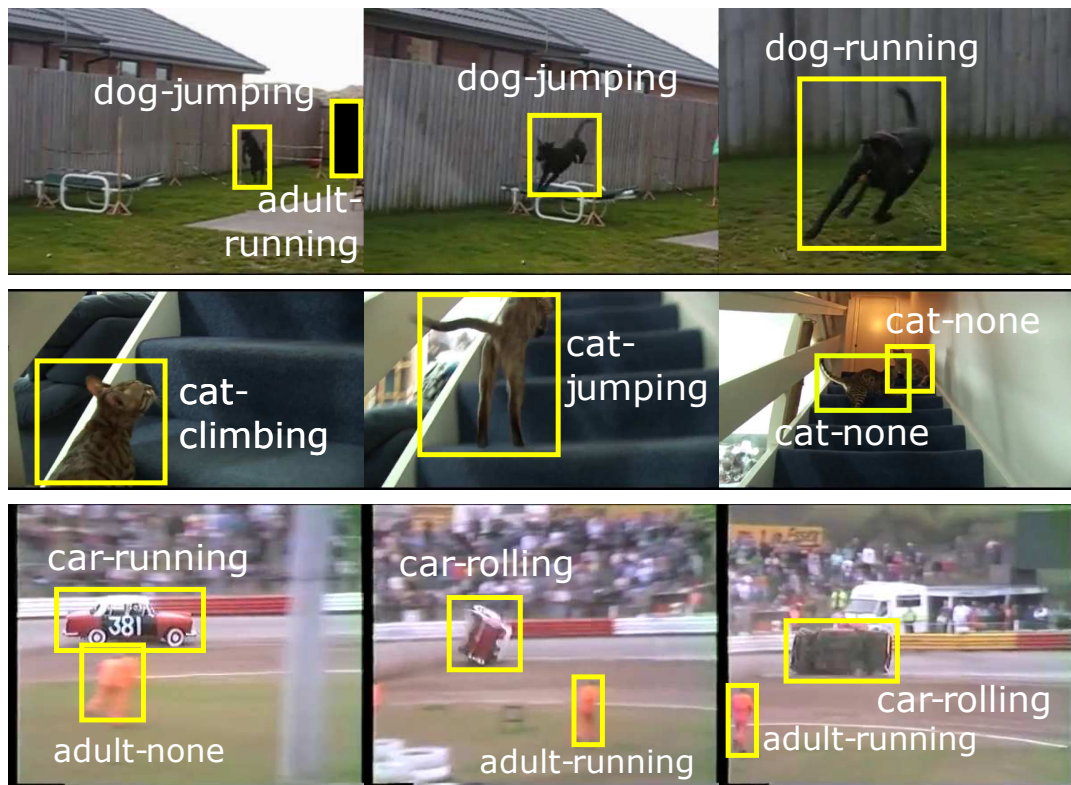


Figure 4.1: Detection examples of different object-action pairs for the videos of the A2D dataset [Xu et al. \[2015\]](#).

object-action detection in Section 4.4. Then, in Section 4.5 we define and explore various adaptation strategies, and present their impact on action localization. We finally draw conclusions in Section 4.6. In Appendix D we report per-class results for the experiments of this chapter.

## 4.1 Overview

Video understanding has received increased attention over the past decade leading to significant advances [[Simonyan and Zisserman, 2014](#); [Venugopalan et al., 2015](#)]. However, most existing approaches focus either on object recognition [[Ren et al., 2015a](#)] or on human action recognition [[Weinzaepfel et al., 2015](#)] separately. To even better understand videos, we need to go beyond these two independent tasks of object recognition and human action recognition and understand the relationship between objects and actions. For instance, an autonomous car should not only be able to detect another *car* (object) or a *human walking* (action), but also a *dog running* or a *ball flying* (object-action) on the street. Other applications include content-based retrieval,

video captioning [Venugopalan et al., 2015; Yao et al., 2015] and health-care robots, for instance helping blind people crossing streets.

Object-action relationships have been studied only on the image domain, where the under question object does not change in time. In videos, however, the visual appearance of the object makes it more challenging to detect its actions and at the same time, the number of possible actions grows exponentially in the number of objects, and there may not be enough training data for each object-action pair [Sadeghi and Farhadi, 2011].

In the first section of this chapter, we propose to jointly detect object-action instances in videos (Figure 4.1). For joint learning of objects and actions, we build an end-to-end two stream network architecture that leverages a multitask objective (Figure 4.2). We compare our proposed end-to-end multitask architecture with alternative ones (Figure 4.3): (i) treating every possible combination of actions and objects as a separate class and (ii) considering a hierarchy of objects-actions: the first level corresponds to objects and the second one to the valid actions for each object (hierarchical).

We show that our method performs as well as these two alternatives while (a) requiring fewer parameters and (b) enabling zero-shot learning of the actions performed by a specific object. More precisely, when training for an object class alone without its actions, our multitask network is able to predict actions for that object class by leveraging actions performed by other objects.

Interestingly, our multitask objective not only allows to effectively detect object-action pairs but also leads to performance improvements on each individual task (i.e., detection of either objects *or* actions). This is because the features learned for one task can help training the other one. We compare to the state of the art for object-action detection on the Actor-Action (A2D) dataset Xu et al. [2015] that contains segmentation annotation for object-action pairs. For a direct comparison we transform our detections into pixelwise segmentation maps by using segmentation proposals Grundmann et al. [2010]; Pinheiro et al. [2016]. Our approach significantly outperforms the state of the art Xu et al. [2015]; Xu and Corso [2016] on this dataset. We finally apply our multitask objective to detect object-action relationships in images on the Visual Relationship Detection (VRD) dataset Lu et al. [2016].

In summary, we make the following contributions:

- We propose an end-to-end *multitask* architecture for joint object-action detection.
- We show that this multitask objective can be leveraged for zero-shot learning of actions.

- We demonstrate the generalization of our multitask architecture by applying it to (a) object-action semantic segmentation and (b) object-action relationships in images.

In Section 4.5, we extend our end-to-end two stream architecture to action localization. We define and explore various late and early fusion strategies that aim at adapting the appearance and motion streams, and we examine their impact on modern action localization benchmarks. Then, we present another fusion adaptation strategy that leverages an adversarial objective and learns a mapping function between the two streams. Exploiting this adversarial objective helps early fusion techniques, as the features coming from both streams are more relevant.

The evaluation code for joint learning of object and action detectors is available at <https://github.com/vkalogeiton/joint-object-action-learning>.

## 4.2 Related Work

Most existing approaches for detection in videos have focused either on object *or* on action localization. Over the past few years, the range of methods covers low-level features [Klaser et al., 2008; Laptev, 2005; Malisiewicz et al., 2011; Prest et al., 2012a; Uijlings et al., 2013; Viola and Jones, 2001; Wang et al., 2015b], structured models that automatically mine mid-level elements [Lan et al., 2015; Ma et al., 2013] or parts [Felzenszwalb et al., 2010; Pandey and Lazebnik, 2011; Raptis et al., 2012] and attributes [Liu et al., 2011]. However, CNNs currently constitute the dominant approach for large-scale and high-quality video detection.

**Object or action detection.** Recent work on object detection [Cinbis et al., 2016; Kang et al., 2016b; Ren et al., 2015a] has shown remarkable progress, mainly thanks to the use of CNNs [Girshick et al., 2014a; Krizhevsky et al., 2012; Liu et al., 2016a]. With Faster R-CNN, Ren et al. [2015a] propose to generate proposals using RPN, that shares convolutional features with the proposal classification branch. For more details about object detectors refer to Section 2.2.

These per-frame detectors are also used in human action localization [Gkioxari and Malik, 2015; Peng and Schmid, 2016; Saha et al., 2016; Weinzaepfel et al., 2015], see Section 2.4. State-of-the-art methods typically use a per-frame detector that provides the spatial information; and then link the detections across time to obtain video-level localizations. To leverage video data, the detector operates on two streams [Simonyan and Zisserman, 2014]: RGB and optical flow. The two streams are trained separately

and the scores are averaged at test time, i.e., late fusion of scores. In contrast, our architecture is a two-stream Faster R-CNN trained end-to-end thanks to a fusion by a fully-connected layer that operates on concatenated features from both streams. It also is trained with a multitask objective that allows us to detect objects *and* actions jointly. Moreover, it is used to test objects alone, actions alone and objects performing actions.

**Joint modeling of objects and actions.** Joint modeling of objects and actions in videos has received little attention so far. For the action localization task, some works [Gupta et al., 2009; Prest et al., 2013] propose to model the interactions of humans and objects. However, the task we tackle in this chapter is significantly different as objects are not used for the actions, but they are the actors. Bojanowski et al. [2013] have considered the case in which different entities can perform a set of actions, but these entities correspond to names of different actors, i.e., to person identification. Closely related to object-action detection are the works on segmenting object-action from Xu and Corso [2016]; Xu et al. [2015]. They use Conditional Random Fields at the supervoxel level to output a semantic segmentation at the pixel level. We show that our detections based on a multitask objective also improve the semantic segmentation performance by leveraging segmentation proposals [Grundmann et al., 2010; Pinheiro et al., 2016].

In images, however, object-action pairs have been modeled implicitly in the context of predicting sentences for images [Mao et al., 2015; Vinyals et al., 2015] and more recently by visual phrases [Sadeghi and Farhadi, 2011] and relationships between objects [Lu et al., 2016]. The task consists in detecting triplets of two objects and their relationship [Lu et al., 2016; Sadeghi and Farhadi, 2011]. Most approaches rely on object detectors. The relationship label is predicted from the bounding box around the two objects, and sometimes from additional modalities such as languages or frequency priors in the training set. We show that our multitask objective allows to predict the relationships between objects without (a) the need to see the whole bounding box and (b) the need to include any priors. In particular, we transform each triplet into two pairs, each consisting of one of the two objects and the interaction. Then, we train our network to detect bounding boxes around objects and also predict an interaction label.

**Zero-shot learning.** Most existing approaches for zero-shot learning of categories rely on attributes [Bucher et al., 2016; Escorcia et al., 2015; Lampert et al., 2014]. Attributes have also been used for human actions [Liu et al., 2011; Yao et al., 2011].

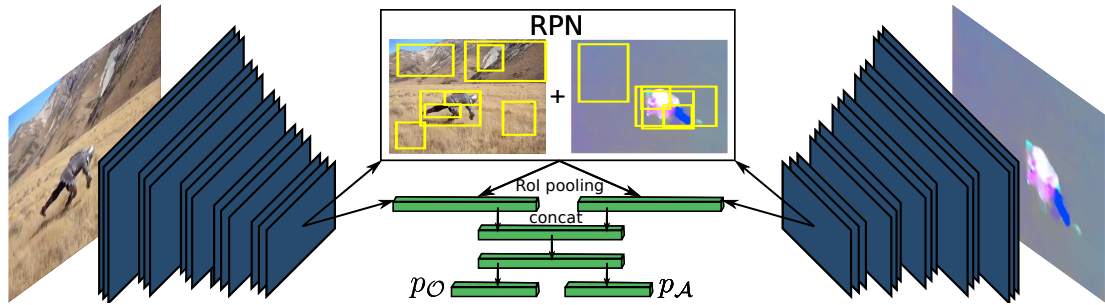


Figure 4.2: Overview of our end-to-end multitask network architecture for joint object-action detection in videos. Blue color represents convolutional layers while green represents fully connected layers. The end-to-end training is done by concatenating the fully connected layers from both streams.

For instance, Liu et al. [2011] were the first to represent actions by sets of attributes. They consider that each action class has an intra-class variability, which they try to model by searching which attributes are relevant for each class. They apply zero-shot learning by manually labeling attributes for all classes, including new ones without visual examples. In contrast, our approach does not require any attribute labels.

### 4.3 End-to-end multitask network architecture for joint learning of objects and actions

Given a video, we aim to detect the objects as well as the actions they are performing. Let  $O$  (resp.  $\mathcal{A}$ ) be the set of objects (resp. actions) labels. Some combinations of actions and objects may not be valid, e.g. *car eating*. We denote by  $\mathcal{V} \subset O \times \mathcal{A}$  the set of valid object-action combinations. In our experiments, we parse the valid object-action combinations from the training set.

#### 4.3.1 End-to-end network architecture

We build an end-to-end two-stream multitask network that proceeds at the frame level (Figure 4.2). As most state-of-the-art methods for object and action detection in videos, we rely on Faster R-CNN [Ren et al., 2015a] and its two-stream variant [Gkioxari and Malik, 2015; Saha et al., 2016; Simonyan and Zisserman, 2014] as it is common in action localization. However, instead of training each stream separately, we propose to fuse both streams, thus enabling effective end-to-end learning. Our end-to-end network



|              | loss                                       | # outputs                 | probability                         | # params |
|--------------|--|---------------------------|-------------------------------------|----------|
| Multitask    | $-\log p_O(o) - \log p_{\mathcal{A}}(a)$   | $ O  +  \mathcal{A}  + 2$ | $p_O(o) \cdot p_{\mathcal{A}}(a)$   | 0.9M     |
| Cartesian    | $-\log p_{\mathcal{V}}(o, a)$              | $ \mathcal{V}  + 1$       | $p_{\mathcal{V}}(o, a)$             | 54.6M    |
| Hierarchical | $-\log p_O(o) - \log p_{\mathcal{A}_o}(a)$ | $ O  +  \mathcal{V}  + 1$ | $p_O(o) \cdot p_{\mathcal{A}_o}(a)$ | 55.4M    |

Table 4.1: Comparison of different losses for object-action learning. We give the number of parameters in the classification layers from the VRD dataset [Lu et al., 2016] where  $|O| = 100, |\mathcal{A}| = 140, |\mathcal{V}| = 13344$  (Section 4.4.4).

has two streams:

1. appearance, which takes as input the RGB data, and
2. motion, which operates on the optical flow [Brox et al., 2004].

Following Gkioxari and Malik [2015], the input of the motion stream is a tensor of three channels with the x and y coordinates of the flow and its magnitude, represented as a 3-channel image. An RPN extracts candidate bounding boxes independently for each stream. We use the set union of the two RPNs and we aggregate features for each candidate box with a RoI pooling layer in each stream. After one fully-connected layer, the two streams are concatenated and fed to another fully-connected layer.

The remaining network layers operate on the fused stream, enabling end-to-end training. This allows us to learn the most relevant features among all possible combinations of appearance and motion. In contrast, late fusion of the softmax probabilities of the two streams [Peng and Schmid, 2016] assumes that both appearance and motion are equally relevant for every class. As we show in Section 4.4.1.1, our proposed fusion significantly outperforms the late fusion.

Finally, we use a multitask loss for detecting objects, actions, and regressing the bounding box coordinates according to the object classes. The total loss  $\mathcal{L}$  of the network is:

$$\mathcal{L} = \mathcal{L}_{\text{RPN}_R} + \mathcal{L}_{\text{RPN}_F} + \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{reg}} \quad , \quad (4.1)$$

with  $\mathcal{L}_{\text{RPN}_R}$  and  $\mathcal{L}_{\text{RPN}_F}$  the losses of the RPN operating on the RGB and flow stream (see Equation 1 in [Ren et al., 2015a]), respectively,  $\mathcal{L}_{\text{cls}}$  the classification loss, i.e., for recognizing objects and actions, and  $\mathcal{L}_{\text{reg}}$  the bounding box regression loss.

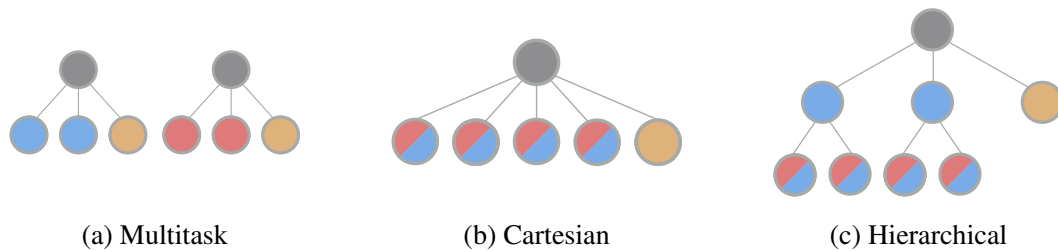


Figure 4.3: *Illustration of the three different ways we consider for jointly learning objects and actions. The blue nodes represent objects and the red ones action classes, while the yellow ones represent the background class.*

### 4.3.2 Joint learning of objects and actions

Given the candidate boxes, the network aims at jointly predicting whether a box contains a particular object and which action this object is performing. Let  $o$  (resp.  $a$ ) be the ground-truth object (resp. action) label of a region proposal in the training set. To classify the boxes, we use a multitask architecture: one component predicts the object class, and a second one predicts the action class, independently of which object is performing it. Besides our proposed multitask architecture, we consider two alternatives to jointly predict object-action pairs: Cartesian product and hierarchy of classes. We now present details for these three objectives. We illustrate them in Figure 4.3 and summarize their main differences in Table 4.1.

**Multitask.** Our multitask architecture relies on a multitask loss, for classifying candidate boxes with both object and action labels. The first branch predicts the object label. It is composed of a fully-connected layer that outputs  $|O| + 1$  scores (one per object class and another one for background) followed by softmax. Let  $p_O$  be the output of this branch. In the same way,  $p_{\mathcal{A}}$  denotes the output of the second branch that predicts the action label, i.e., of dimension  $|\mathcal{A}| + 1$ . We use a log loss on both object and action classification:

$$\mathcal{L}_{\text{cls}}^{\text{Multitask}} = -\log p_O(o) - \log p_{\mathcal{A}}(a) . \quad (4.2)$$

This version uses  $|O| + |\mathcal{A}| + 2$  outputs (Figure 4.3 (a)). For  $|O| = 100$  and  $|\mathcal{A}| = 140$  the number of parameters in the classification layers is 0.9M (VRD dataset [Lu et al., 2016] used in Section 4.4.4). At test time, the probability of a box to be the object-action instance  $(o, a)$  is given by  $p_O(o) \cdot p_{\mathcal{A}}(a)$ .

**Cartesian product.** Another solution is to consider each object-action pair as a separate class, e.g. *bird flying* (Figure 4.3 (b)). In this case, there is only one branch for classification with  $|\mathcal{V}| + 1$  outputs. We denote as  $p_{\mathcal{V}}$  the output of this branch. The classification loss is:

$$\mathcal{L}_{\text{cls}}^{\text{Cartesian}} = -\log p_{\mathcal{V}}(o, a) . \quad (4.3)$$

This version uses  $|\mathcal{V}| + 1$  outputs, which is in the order of  $|\mathcal{A}| \times |O|$ . For instance, for  $|\mathcal{V}| = 13344$  (VRD dataset [Lu et al., 2016]) the number of parameters in the classification layer is 54.6M, i.e.,  $50\times$  more than in the multitask (Table 4.1). This makes it less scalable than our multitask objective and does not allow sharing of action labels across object classes, which is required for zero-shot learning.

In the multitask case, samples of an object-action pair help training the detector of this object, which in turn helps detecting it doing other actions; e.g. *adult-running* and *adult-walking* samples help improving the *adult* detector. In contrast, by using the Cartesian product, each training sample helps training only one particular object-action detector. At test time, the probability of being an object-action instance  $(o, a)$  is given by  $p_{\mathcal{V}}(o, a)$ .

**Hierarchy of classes.** We also consider the set of valid object-action classes as a hierarchy (Figure 4.3 (c)). The first branch  $p_O$  predicts the object. For each object  $o$ , any branch  $p_{\mathcal{A}_o}$  predicts the actions among the valid ones  $\mathcal{A}_o$  for  $o$ . In this case, the classification loss is:

$$\mathcal{L}_{\text{cls}}^{\text{Hierarchy}} = -\log p_O(o) - \log p_{\mathcal{A}_o}(a) . \quad (4.4)$$

This version uses a total of  $|O| + 1$  outputs for the first level and  $|\mathcal{V}|$  for the second level, see Figure 4.3 (c). For instance, for  $|O| = 100$  and  $|\mathcal{V}| = 13344$  the number of parameters in the classification layers is 55.4M, i.e.,  $50\times$  more than in the multitask (Table 4.1). At test time, the probability of being an object-action instance  $(o, a)$  is given by  $p_O(o) \cdot p_{\mathcal{A}_o}(a)$ .

**Per-object regression.** In all cases, we refine the proposal output by the RPN using a per-object regression of the bounding box coordinates. The RPN minimizes the geometric difference between the proposals and the ground-truth boxes. We follow [Ren et al., 2015a] and make the regression target scale-invariant by normalizing it by the

size of the proposal. By using a per-object regression, we obtain the following regression loss:

$$\mathcal{L}_{\text{reg}} = \text{Smooth-L1}(u_o - t_{o,a}) , \quad (4.5)$$

with  $u_o$  the output (four coordinates) of the regression branch  $u$  corresponding to object  $o$ , with  $t_{o,a}$  the regression target (four coordinates) for a proposal that covers an object  $o$ , and:

$$\text{Smooth-L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1, \\ |x| - 0.5 & \text{otherwise.} \end{cases} \quad (4.6)$$

More details for the regression loss are available in [Ren et al., 2015a].

## 4.4 Experimental results for multitask learning

In this section, we study the impact of each of our contributions separately. We first examine joint detection of objects and actions (Section 4.4.1) and zero-shot learning (Section 4.4.2). Next, we compare our proposed multitask architecture to the state of the art on semantic segmentation of object-action pairs (Section 4.4.3) and relationship detection in images (Section 4.4.4). Note that the per-class results of this section can be found in AppendixD.

**Implementation details.** Our framework is based on Faster R-CNN [Ren et al., 2015a], using the VGG16 model of Simonyan and Zisserman [2015] as the underlying CNN architecture. We initialize both streams using pre-training on ILSVRC 2012 classification challenge [Krizhevsky et al., 2012]. This is in line with Wang et al. [2016b], which shows that pre-training on ILSVRC 2012 instead of UCF-101 [Soomro et al., 2012] improves video classification accuracy.

### 4.4.1 Joint detection of objects and actions in videos

In this section, we evaluate our proposed end-to-end architecture for joint detection of object-action pairs. We start by validating the effectiveness of our end-to-end network (Section 4.4.1.1) and then, we examine the joint learning with the multitask objective (Section 4.4.1.2).

| information / datasets |          | A2D | YTO  | VID  |
|------------------------|----------|-----|------|------|
| objects                |          | ✓   | ✓    | ✓    |
| actions                |          | ✓   | -    | -    |
| # videos               | training | 3K  | 106  | 3,9K |
|                        | test     | 746 | 49   | 555  |
| # annotations          | training | 16K | 4K   | 1,7M |
|                        | test     | 4K  | 2,5K | 170K |

Table 4.2: Overview of the video datasets used in our experiments.

**Video datasets.** Table 4.2 shows some statistics of the datasets we use. For object-action detection we use the Actor-Action (A2D) dataset [Xu et al., 2015], which has sparse frame-level annotations for both objects and actions in videos. To the best of our knowledge, it is the only video dataset with bounding box and semantic segmentation annotations for object-action pairs. It contains 7 objects (*adult*, *baby*, *ball*, *bird*, *car*, *cat*, and *dog*) performing 8 different actions (*climb*, *crawl*, *eat*, *fly*, *jump*, *roll*, *run*, *walk*) or no action.

We also use two video datasets only for object detection: the YouTube-Objects (YTO) dataset [Kalogeiton et al., 2016; Prest et al., 2012a] and the ‘object detection in video’ (VID) track of the ILSVRC [Russakovsky et al., 2015b]. YTO consists of videos collected from YouTube with 10 classes of moving objects, e.g. *aeroplane*, *car*. VID contains bounding boxes for 30 object classes including rigid objects, e.g. *motorcycle*, *watercraft*, and animals, e.g. *fox*, *monkey*. For more details about these two datasets refer to Section 3.3.

**Protocol.** We measure the detection performance using the PASCAL VOC protocol [Everingham et al., 2007, 2010]: a detection is correct if its intersection-over-union overlap (IoU) with a ground-truth box is greater than 0.5 and its labels (object and action) are correctly predicted. The performance for a class is the average precision (AP), and the overall performance is captured by the mean over all classes (mAP).

#### 4.4.1.1 End-to-end architecture

We want to quantify the effectiveness of our proposed end-to-end architecture that consists of two streams fused (a) at the proposal (RoI) level and (b) at the feature level (Figure 4.2). We evaluate the impact of fusion for *object* detection alone. We

| input |      | RoI |      | Stream      | A2D         | YTO         | VID         |
|-------|------|-----|------|-------------|-------------|-------------|-------------|
| RGB   | Flow | RGB | Flow | Fusion      |             |             |             |
| ✓     | -    | ✓   | -    | -           | 63.1        | 58.9        | 45.2        |
| -     | ✓    | -   | ✓    | -           | 32.0        | 32.3        | 5.0         |
| ✓     | ✓    | ✓   | ✓    | late        | 61.6        | 57.3        | 33.9        |
| ✓     | ✓    | ✓   | ✓    | <b>ours</b> | <b>65.3</b> | <b>62.2</b> | <b>48.1</b> |

Table 4.3: *Impact of end-to-end training: mAP for object detection of different training scenarios on the A2D, YTO and VID datasets.*

perform experiments on the three video detection video datasets (A2D, YTO and VID). Table 4.3 shows all the mAP results for the different cases we consider.

**Impact of RGB and Flow cues.** To examine the impact of the RGB and flow cues, we train each stream separately. The first two rows of Table 4.3 show that the RGB stream significantly outperforms the flow one. This is due to the fact that the RGB stream is able to learn information about how the objects look, which is a distinctive cue across different object classes. The flow stream performs worse than the RGB one in general, and is particularly poor on the VID dataset. This is because most objects in VID move only slightly, or their motion is not discriminative for the class.

**Impact of end-to-end training.** Our proposed fusion of the two streams enables end-to-end training. We examine the impact by comparing our proposed fusion of streams with late fusion of scores [Gkioxari and Malik, 2015; Peng and Schmid, 2016] (Section 4.2). In the latter, i.e., late fusion of scores, we train the two-stream network fusing only the region-proposal layers and then average the classification scores of each stream as [Gkioxari and Malik, 2015; Saha et al., 2016]. Results in Table 4.3 show that, for all video datasets, using late score fusion reduces the detection performance compared to using the RGB stream alone.

Interestingly, this is opposite of the findings in human action localization Gkioxari and Malik [2015]; Peng and Schmid [2016], where performance increases due to the the significance of motion cues for actions. This shows that the two-stream architecture cannot be used as it is for object detection in videos and highlights a clear difference between object and human action detection. In contrast, on all object detection datasets, our proposed fusion outperforms the other cases: it leads to an increment over the late score fusion of approximately 2-3%. This shows that the network successfully

| training     | test on |         |                   |
|--------------|---------|---------|-------------------|
|              | objects | actions | objects + actions |
| objects      | 65.3    | -       | -                 |
| actions      | -       | 56.2    | -                 |
| Baseline     | -       | -       | 43.1              |
| Cartesian    | 67.2    | 60.2    | 49.2              |
| Hierarchical | 67.9    | 59.6    | 49.6              |
| Multitask    | 68.3    | 60.0    | 48.9              |

Table 4.4: *mAP* of six different models when training with objects (first row), actions (second row), when multiplying their scores (third row) or when jointly training with objects and actions (last three rows) on A2D.

learns when to leverage motion information and more importantly, how to jointly learn features coming from the two stream.

#### 4.4.1.2 Multitask learning

In this section, we evaluate our proposed multitask learning of objects and actions. We start by evaluating the performance only on *object* or on *action* detection. Therefore, we train and test our network with only object *or* only action labels (first two rows of Table 4.4). We also compute a baseline (third row of Table 4.4) for object-action detection in which we combine the object and the action detector trained separately. More precisely, for each object detection, we obtain object-action scores by multiplying the object scores with the action scores from the most overlapping action box.

Table 4.4 also reports the results of our proposed multitask architecture trained with objects and actions from the A2D dataset. The most interesting finding is that our multitask training improves the performance on each task separately (Table 4.4 objects, actions and multitask rows). In particular, when testing just on objects (68.3%) or just on actions (60.0%), our joint training outperforms training alone with objects (65.3%) or with actions (56.2%). The reasons are that the multitask network is (a) better able to generalize, (b) less prone to overfit to the training samples and (c) benefits from sharing examples across classes.

We also consider two alternative ways to jointly detect objects and actions (Section 4.3.2 and Figure 4.3): (a) Cartesian product of object-action labels and (b) hierarchy of object-action classes. Table 4.4 (Cartesian and hierarchical) reports the results



when we train these two networks on the A2D dataset. We observe that they both perform similarly to our multitask network. All three networks have the advantage of being able to distinguish different ways objects perform each action (Table 4.1).

**Discussion.** In practice there are similarities in the way different objects perform the same action (e.g. *dog* and *cat eating*) and in the way the same object performs different actions (e.g. *dog walking* and *running*).

Thus, our multitask objective allows the network to exploit the commonality among the two tasks, and hence, what is learned for each task facilitates the learning of the other. In a nutshell, our multitask architecture is a simpler model, able to reach the same performance as the alternative architectures while requiring much fewer parameters (Table 4.1 # params) and enabling zero-shot learning (Section 4.4.2). For instance, in Section 4.4.4 we clearly show the benefit of our multitask architecture compared to the Cartesian and hierarchical architectures for a large number of objects and actions due to its lower number of parameters.

Note that both losses (object and action) contribute equally to the overall loss (Equation 4.2), as they are of the same type (softmax), and the tasks they address are of the same difficulty. To validate this, we vary the weight of the action loss over 0.5, 1, 2 and observe insignificant variations ( $< 0.5\%$ ) in the object-action mAP on A2D.

#### 4.4.2 Zero-shot learning of actions

An important advantage of our end-to-end multitask architecture is its capability of predicting actions for an object without having trained for these particular object-actions combinations. To validate this intuition, we experiment on the A2D dataset (Table 4.2), which contains annotations for 7 objects performing 8 different actions in videos (and an extra action class ‘none’ when the object does not perform one of the 8 defined actions). We train the network seven times, where each time we remove for one object  $o'$  all its action labels. For instance, we remove all action labels for the object *cat*, but keep the *cat* examples for training the object detector.

Equation 4.2 is replaced by:

$$\mathcal{L}_{\text{cls zero-shot}}^{\text{Multitask}} = -\log p_O(o) - [o' \neq o] \log p_{\mathcal{A}}(a) . \quad (4.7)$$

Note that the object classifier is not changed, while the action classifier is learned

|       | climbing    | crawling    | eating      | flying     | jumping     | rolling     | running     | walking     | none        | avg.        |
|-------|-------------|-------------|-------------|------------|-------------|-------------|-------------|-------------|-------------|-------------|
| adult | 7.0 (78.2)  | 7.8 (72.5)  | 19.6 (80.0) | -          | 11.0 (43.7) | 24.3 (50.7) | 6.3 (55.2)  | 13.6 (58.8) | 33.3 (45.8) | 15.3 (60.1) |
| baby  | 17.1 (63.1) | 31.7 (76.4) | -           | -          | -           | 33.2 (85.4) | -           | 39.1 (77.9) | 7.1 (31.9)  | 25.6 (64.9) |
| ball  | -           | -           | -           | 0.4 (19.3) | 3.5 (29.8)  | 10.7 (42.2) | -           | -           | 8.0 (11.1)  | 5.6 (28.0)  |
| bird  | 16.8 (51.8) | -           | 13.4 (38.0) | 9.0 (66.2) | 6.4 (32.3)  | 28.6 (60.2) | -           | 7.7 (55.0)  | 2.4 ( 2.3)  | 12.1 (43.3) |
| car   | -           | -           | -           | 8.8 (42.2) | 1.5 (90.5)  | 36.5 (66.8) | 2.7 (63.8)  | -           | 5.1 (17.4)  | 10.9 (55.9) |
| cat   | 32.3 (60.2) | -           | 28.9 (58.6) | -          | 9.6 (21.7)  | 43.8 (68.2) | 8.0 (31.0)  | 19.1 (49.2) | 3.1 ( 5.8)  | 20.7 (43.7) |
| dog   | -           | 7.9 (58.2)  | 47.3 (74.2) | -          | 17.9 (41.6) | 25.5 (38.5) | 10.3 (31.4) | 34.0 (67.2) | 1.8 ( 5.3)  | 20.7 (42.3) |

Table 4.5: Evaluation of zero-shot learning for object-action pairs on A2D. For each object, we report the AP when excluding all actions of this object at training. The numbers in parenthesis indicate the AP when training with all object-action pairs.

only on the actions performed by the objects different from  $o'$ . This approach to zero-shot learning does not assume any prior knowledge such as attributes of the unseen classes Liu et al. [2011].

We report the results of zero-shot learning in Table 4.5. We also report the AP when training with all object-action pairs. The results show that our network is able to infer information about actions not seen at training time for a given object. We observe that there are some object-action pair for which the AP is only slightly decreased, e.g. *cat rolling* or *dog eating*. This is because these objects share commonalities with others, e.g. *cat* and *dog eating*. In contrast, we observe poor performance for objects like *ball* which do not share similarities with other objects of the dataset. For object classes that share similarities in actions, such as *cat* and *dog*, our multitask architecture outperforms chance level classification of unknown actions by a large margin (+15%), while for classes that do not share commonalities with other classes, like *adult* the gain is smaller (+5%). Overall, our multitask architecture outperforms the chance level classification.

### 4.4.3 Object-action segmentation

A2D comes with annotations for semantic segmentation of object-action pairs (Table 4.2). In this section, we extend our bounding box detections to pixelwise segmentation and we compare our results to the state of the art.

**Metrics.** Following [Xu et al., 2015], we measure class-average pixel accuracy and global pixel accuracy. Accuracy is the percentage of pixels for which the label is correctly predicted, either over all pixels (global) or first computed for each class separately and then averaged over classes (class-average). We also evaluate our segmen-

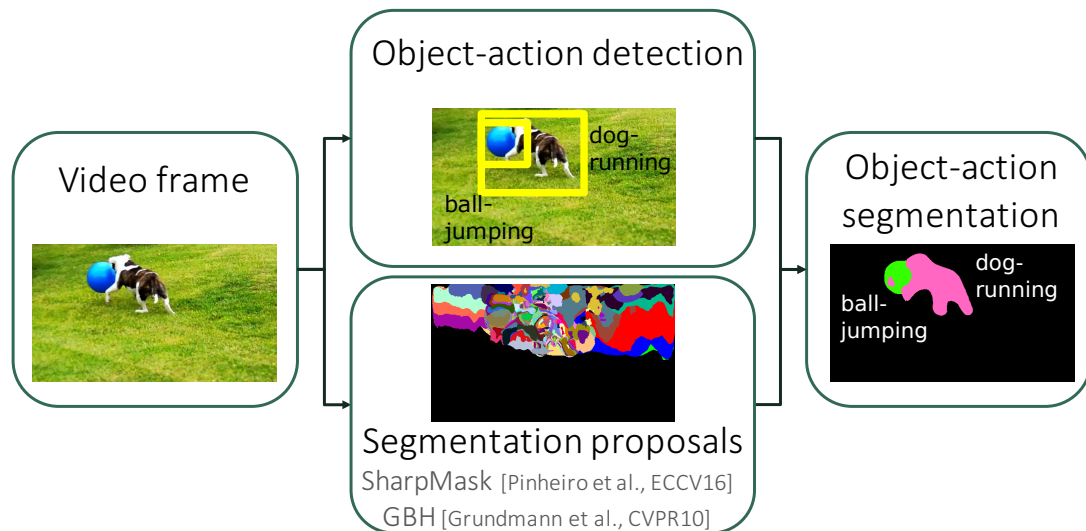


Figure 4.4: Overview of our setup for extending our object-action detections to semantic segmentations using segmentation proposals from [Grundmann et al., 2010; Pinheiro et al., 2016]. For each bounding-box detection, e.g. dog running we find the segmentation proposals that overlaps mostly with the detection and use it as the final segmentation.

tations using mIoU, i.e., the IoU between the ground-truth segmentation and output segmentation averaged over all classes. mIoU is better suited as it is not biased towards background which is the most present class and it penalizes errors when too many pixels are set to a particular label instead of background.

**Setup.** Our two-stream multitask model predicts bounding boxes for each object-action pair. We extend our detections to pixelwise segmentations of object-action pairs by using segmentation proposals from either (a) the recently proposed SharpMask [Pinheiro et al., 2016] or (b) the hierarchical video segmentation method GBH by Grundmann et al. [2010], which is the one used by the state-of-the-art GPM method [Xu and Corso, 2016]. Figure 4.4 shows a schematic representation of our setup. For each frame, we first apply non-maximum suppression on the bounding-box detections that have a score greater than 0.5. In that way, we discard most of the irrelevant (low confidence) bounding-box detections and we keep only a few for each frame. Then, for each detection, we select the segmentation proposal that overlaps the most with it (according to IoU). If there is no such proposal, we directly use the rectangular detection itself as a segmentation mask. While our setup is simple, it serves as a baseline to evaluate our detections for semantic segmentation.

| methods                        | object      |             |             | action      |             |             | object + action |             |             |
|--------------------------------|-------------|-------------|-------------|-------------|-------------|-------------|-----------------|-------------|-------------|
|                                | ave         | glo         | mIoU        | ave         | glo         | mIoU        | ave             | glo         | mIoU        |
| Trilayer [Xu et al., 2015]     | 45.7        | 74.6        | -           | 47.0        | 74.6        | -           | 25.4            | 76.2        | -           |
| GPM (TSP) [Xu and Corso, 2016] | 58.3        | 85.2        | 33.4        | 60.5        | 85.3        | 32.0        | 43.3            | 84.2        | 19.9        |
| GPM (GBH) [Xu and Corso, 2016] | 59.4        | 84.8        | 33.3        | 61.2        | 84.9        | 31.9        | 43.9            | 83.8        | 19.9        |
| <b>Ours (GBH)</b>              | 72.9        | 85.8        | 42.7        | <b>61.4</b> | 84.6        | 35.5        | <b>48.0</b>     | 83.9        | 24.9        |
| <b>Ours (SharpMask)</b>        | <b>73.7</b> | <b>90.6</b> | <b>49.5</b> | 60.5        | <b>89.3</b> | <b>42.2</b> | 47.5            | <b>88.7</b> | <b>29.7</b> |

Table 4.6: Comparison to the state of the art for object, action and object-action segmentation on A2D using class-average pixel accuracy (ave), global pixel accuracy (glo) and mean Intersection over Union (mIoU) metrics.

**Results.** The first three rows of Figure 4.5 show correctly labeled and segmented object-action pairs. We observe that our segmentation results are accurate, even in difficult cases, such as small objects (e.g. *birds*) or cluttered scenes (e.g. *adults running*). The two last rows show typical failure cases. In the fourth row, the action label of one *adult* is incorrect and there are some detections considered as wrong due to missing annotations. In the last row we miss the *adult* for which only one arm is visible.

Table 4.6 provides a quantitative comparison between our results and the state of the art [Xu et al., 2015; Xu and Corso, 2016] on A2D. In particular, it reports the three metrics (average, global and mIoU) when testing on (a) objects alone, i.e., percentage of the pixels of each object with the correct *object* label, (b) actions alone, i.e., percentage of the pixels of each object with the correct *action* label, and (c) both objects and actions, i.e., percentage of the pixels of each object with correct both *object* and *action* labels. When using SharpMask, we outperform the previous state of the art for all metrics and all tasks, except for average accuracy on action segmentation, where we match [Xu and Corso, 2016]. Our improvements are particularly significant for object segmentation (+14% class-average accuracy, +16% mIoU) and joint object and action segmentation (more than +5% on all metrics). Note that we do not use any training segmentation from the A2D dataset (SharpMask is pre-trained on MS COCO Lin et al. [2014b]). Furthermore, we observe that even when using the same underlying method (GBH [Grundmann et al., 2010]), we perform on par or better than [Xu et al., 2015; Xu and Corso, 2016] in all metric-task combinations.



Figure 4.5: Examples of semantic segmentation with (from left to right): the frame, the ground-truth and the segmentation output obtained when combining our approach with proposals from SharpMask [Pinheiro et al. \[2016\]](#). The colors of the segmentations represent an object-action pair. Note that we do not use any object-action segmentation at training time.

#### 4.4.4 Relationship detection of objects and actions

In this section we use only images, and therefore we use only the RGB stream as there is no flow for images. We apply our model to visual relationship detection, where we detect relationships between objects, defined as triples: object1 - interaction - object2. To do so, we transform each triplet into two pairs, each consisting of an object and an interaction and use them to train our multitask architecture.

| Modality | Method                                   | Phrase detection |      | Relationship detection |      |
|----------|--|------------------|------|------------------------|------|
|          |  | R@100            | R@50 | R@100                  | R@50 |
| V        | VP [Sadeghi and Farhadi, 2011]           | 0.07             | 0.04 | -                      | -    |
|          | Joint CNN [Simonyan and Zisserman, 2014] | 0.09             | 0.07 | 0.09                   | 0.07 |
|          | VRD [Lu et al., 2016]                    | 2.6              | 2.2  | 1.9                    | 1.6  |
|          | Baseline                                 | 11.9             | 7.7  | 7.1                    | 4.5  |
|          | <b>Ours Multitask</b>                    | 18.3             | 14.5 | 11.3                   | 8.6  |
| V+L+F    | VRD [Lu et al., 2016]                    | 17.0             | 16.2 | 14.7                   | 13.9 |

Table 4.7: Comparison of our multitask model to baselines and to the state-of-the art visual relationships results on the VRD dataset for phrases and relationship detection. We report  $R@100$  and  $R@50$  for methods using only visual cue (V) or also language and frequency priors (V+L+F).

**Dataset and protocol.** We employ the Visual Relationship Detection (VRD) dataset [Lu et al., 2016] that examines object relationships. It contains 4k training and 1k test images with 38k relationships between objects, such as *person kick ball*, *person wear shirt*, *motorcycle has wheel*. There are 100 different objects and 70 interaction types.

We consider here visual phrase detection [Sadeghi and Farhadi, 2011], where the goal is to output a triplet object1 - interaction - object2 and localize it with one box having an IoU over 0.5 with the ground-truth box. In addition to phrase detection, we also evaluate relationship detection: the task consists in detecting a triplet object1 - interaction - object2 with two bounding boxes on object1 and object2, both having an IoU over 0.5 with their ground-truth boxes.

For evaluation, the metric used is recall @100 and recall @50 (denoted as  $R@N$ ) and not mAP, as not all possible interactions are annotated in the test images. In each image, the top  $N$  detections are kept and recall is measured.

**Model.** To detect relationships using our multitask architecture, we transform each object1-interaction-object2 triplet into two pairs, each consisting of an object and an interaction label. More precisely, we double the set of all possible interactions, by including their passive forms. For example, the triplet *human kicks ball* becomes two pairs: (i) one with object *human* and action *kick*, and (ii) another pair with object *ball* and action  $\widetilde{kick} = \textit{being kicked}$ . In that way, our training set consists of 100 object classes performing 140 different actions. Note here that the possible number of outputs is  $100 + 140 + 2$  for our multitask objective.

At test time, we keep all detection with score over 0.5 and apply non-maximum



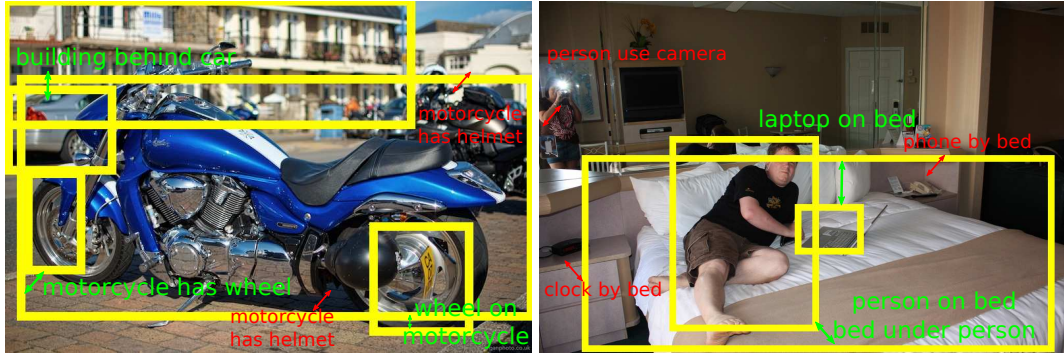


Figure 4.6: Qualitative object-action relationship results on the VRD dataset. The yellow color depicts our correct boxes with their green label, while the red color represents missed interactions until  $R@100$ .

suppression. For each pair of object detections, we score each possible interaction using the multiplication of the object scores and the interaction score. The interaction score is defined as the combination of the score of an interaction from the first object and its passive form from the second object, i.e., the interaction score of *kick* in *human kicks ball* includes both scores of *kick* for the *human* and *being kicked* for the *ball*.

**Results.** Table 4.7 reports the  $R@100$  and  $R@50$  for the two tasks we examine, i.e., phrase and relationship detection. We outperform all previous state-of-the-art results on both tasks and at both operating points, when comparing to methods based purely on the images (Lu et al. [2016]; Sadeghi and Farhadi [2011]; Simonyan and Zisserman [2014]). Moreover, our results are only a little worse than those of Lu et al. [2016], where they enhance their visual model with some frequency prior as well as language priors by leveraging the semantic similarities of relationships in term of words. In particular, we perform on par on phrase detection (+1% at  $R@100$  and  $-2%$  at  $R@50$ ). Note how our method features a clear increment from  $R@50$  to  $R@100$ , which shows its potential to correctly detect interactions that may be lower in the recall list. Hence, including some language or spatial priors could significantly increase our performance. Figure 4.6 shows some qualitative results.

**Benefits of the multitask training.** We compare our multitask architecture with a baseline approach where we multiply the scores of two separate networks, one trained on objects and another one trained on interactions. Table 4.7 shows that our multitask architecture outperforms this alternative (‘Baseline’ row). This comparison highlights the benefit of joint training compared to training for each task separately. We have

| Modality | Method                | Phrase detection |      | Relationship detection |      |
|----------|-----------------------|------------------|------|------------------------|------|
|          |                       | R@100            | R@50 | R@100                  | R@50 |
| V        | VRD [Lu et al., 2016] | 1.1              | 0.8  | 0.8                    | 0.7  |
|          | Baseline              | 4.3              | 2.3  | 2.4                    | 1.3  |
|          | <b>Ours Multitask</b> | 5.5              | 3.4  | 2.9                    | 1.9  |
| V+L+F    | VRD [Lu et al., 2016] | 3.8              | 3.4  | 3.5                    | 3.1  |

Table 4.8: Comparison of our multitask model to the state-of-the-art methods for zero-shot detection of visual relationships on the VRD dataset. We report R@100 and R@50 for methods using only visual cue (V) or also language and frequency priors (V+L+F).

also evaluated the Cartesian and hierarchical combination of objects and actions (Section 4.3.2) and found that they perform poorly (for both R@100 is around 0%). This can be explained by lack of training data necessary to determine the large number of parameters (55M in Table 4.1).

**Zero shot learning.** The test set of the VRD dataset contains 1.9k triplets that never occur in the training set. Our architecture allows zero-shot learning and we report the results on these triplets in Table 4.8. Our method outperforms the state-of-the-art method Lu et al. [2016] when using only the visual modality (no language or frequency prior). Additionally, for phrase detection we detect unseen-at-training interactions better than Lu et al. [2016], even when they also use language and frequency priors. Finally, our multitask architecture outperforms the baseline by a significant margin, highlighting the benefit of joint training compared to separate one.

## 4.5 Exploring action adaptation techniques for action localization

In this section, we explore an action adversarial adaptation technique for action localization in videos. Modern action localization methods [Gkioxari and Malik, 2015; Saha et al., 2016; Simonyan and Zisserman, 2014] are based on the two-stream variant of Faster R-CNN of Ren et al. [2015a]: (a) an appearance stream, which takes as input RGB frames  $X_S$  and (b) a motion stream, which operates on optical flow  $X_T$ . The drawback of these methods [Peng and Schmid, 2016] is that they combine the two streams by averaging their softmax probabilities, i.e., late fusion. Instead, in the pre-



vious section we propose an early fusion mechanism that enables end-to-end training and testing, see Section 4.3. Early fusion, however, is not trivial: RGB and flow come from different domains and, hence, their representation is different [Gupta et al., 2016; Hoffman et al., 2016].

Therefore, in Section 4.5.1 we explore different strategies for fusing the two streams together with multi-scale training and testing and in Section 4.5.2 we examine their impact on action localization. Then, in Section 4.5.3 we learn a common mapping of the features of the RGB and flow frames based on an adversarial objective, aiming at improving action localization and we report the results in Section 4.5.4.

**Implementation details.** As in Section 4.4, in this section our framework is based on Faster R-CNN [Ren et al., 2015a], using the VGG16 Simonyan and Zisserman [2015] architecture. We initialize both streams using the pre-trained ILSVRC 2012 model [Krizhevsky et al., 2012].

**Datasets.** We experiment on three action localization datasets: UCF-Sports, J-HMDB, UCF-101, see Section 2.4.4.

**Metrics.** We measure the detection performance only at the frame level, as we want to examine the quality of the detections independently of any linking strategy. We report frame-mAP, as described in paragraph Protocol in Section 4.4.1. For each class, we compute the average precision (AP) and report the average over all classes.

**Outline.** In Section 4.5.1, we first define and explore different action adaptation strategies, and then in Section 4.5.2 we examine their impact on action localization. Then, Section 4.5.3 presents another action adaptation strategy based on an adversarial objective, and finally Section 4.5.4 shows the experimental results of the adversarial adaptation for action localization.

### 4.5.1 Action adaptation

In this section, we explore alternative strategies for fusing the appearance and flow streams. In all cases, we consider two-stream end-to-end networks, where the two streams are fused at the Region Proposal Network (RPN) level, where the RPN of each stream extracts candidate bounding boxes and then, in each stream, we aggregate features for all candidate boxes with a Region-of-Interest (RoI) pooling layer.

Given the RPN fusion, we consider two alternatives for fusing the two streams:

- i. *union* fusion, where we consider the set union of the outputs from both streams: boxes coming from the fused RPN that are regressed and scored both from the RGB stream and from the flow stream (presented in [Singh et al., 2017]).
- ii. *late* fusion, where for each anchor box we average the scores from both streams and keep the regressed box from the stream where the anchor box was generated.

Additionally, we also consider the case of early fusion  $ef$ , as described in Section 4.3.1. Our end-to-end network consists of two streams that are fused in two different locations: (a) at the RPN level, and (b) at the convolutional or fully-connected layers level, where after  $n$  layers, the features of both streams are concatenated and fed to  $r$  fully-connected layers that operate on the fused stream.

We vary the early fusion location  $n$ , the number of remaining fully-connected layers  $r$  and feature fusion operation. As Figure 4.7 shows, we explore  $n = 5$  early fusion locations:

- iii.  $ef_1$ : before the convolutional layer conv1, where we fuse the input images to a common embedding, and hence the rest of the network operates on one fused stream.
- iv.  $ef_5$ : after the convolutional layer conv5, where we fuse the responses from the conv5 layers before the RPN, and therefore the whole network (including the RPN) operates on one fused stream.
- v.  $ef_6$ : after the fully-connected layer fc6, where we fuse the RPNs from both streams, and we also concatenate the feature vectors of each stream after the fc6 layer. Fusing the RPNs means using the set union of the two RPNs for the remaining layers. Note that this is the case described in our end-to-end network in Section 4.3.
- vi.  $\tilde{ef}_6$ : after the fully-connected layer fc6, where we fuse the RPNs from both streams, and we also sum the feature vectors of each stream after the fc6 layer. This case is the same as v. except for the function to fuse the fc layers.
- vii.  $\tilde{ef}_8$ : after the fully-connected layer fc8, where we fuse the RPNs from both streams, and also for each anchor box we keep the regression coming from the RGB stream, and sum its softmax probabilities coming from both streams after

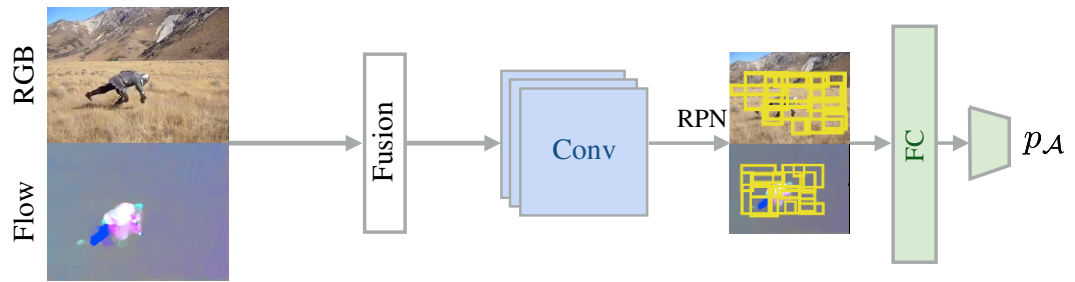
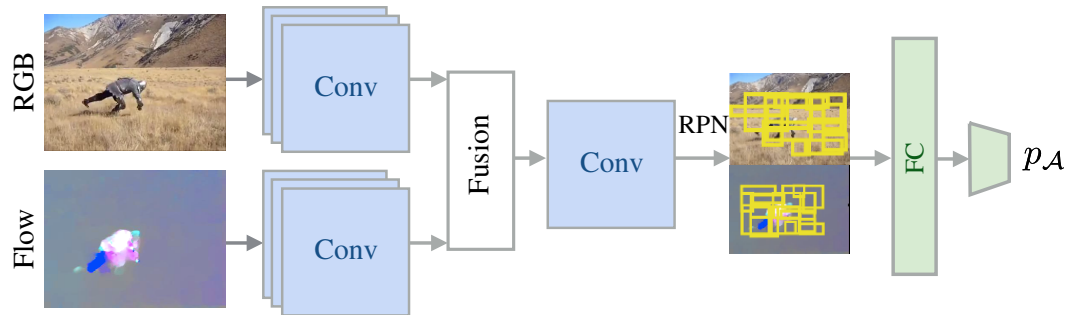
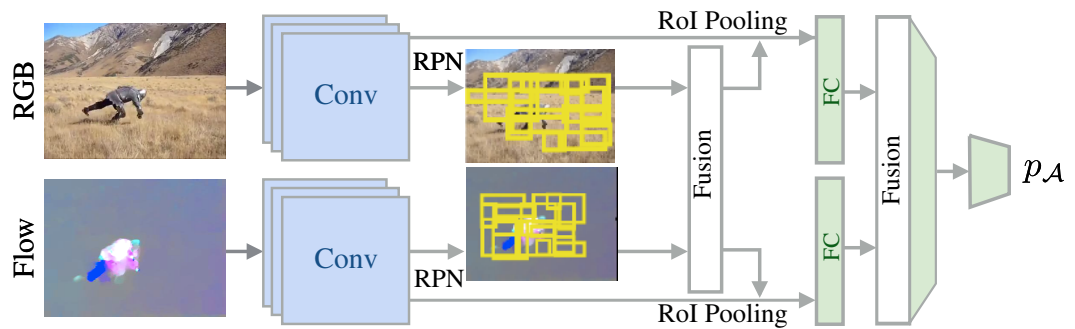
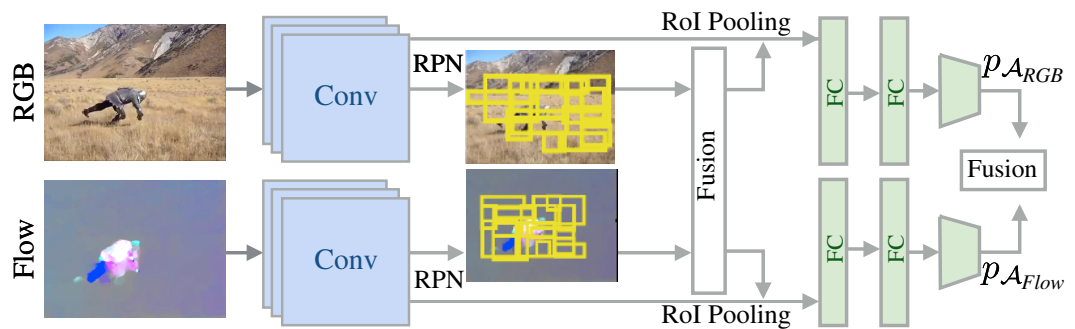
(a) iii.  $ef_1$ : fusing at the  $conv1$  layer(b) iv.  $ef_5$ : fusing at the  $conv5$  layer(c) v.-vi.  $ef_6$  and  $\tilde{ef}_6$ : RPN and  $fc6$  fusion (concatenation and summation)(d) vii.  $ef_8$ : RPN and softmax fusion.

Figure 4.7: Illustration of the five different early fusion strategies: fusing at the (a)  $conv1$ , (b)  $conv5$ , (c)  $fc6$ , and (d) softmax layer.

the fc8 layer. In other words, this operation is equivalent to fusing the softmax probabilities of each box. We keep the regressed boxes from the RGB stream, as appearance is more relevant for regressing boxes, in particular for actions with limited motion.

Respectively, the remaining layers that operate on the fused stream are: iii.  $r = 8$ : the whole network, iv.  $r = 3$ : the fully-connected layers fc6-fc8, v.  $r = 2$ : the fully-connected layers fc7-fc8, vi.  $r = 2$ : the fully-connected layers fc7-fc8, and vii.  $r = 0$ : none.

The training loss is defined by Equation 4.1. The classification loss is:

$$\mathcal{L}_{\text{cls}} = -\log p_{\mathcal{A}} \quad , \quad (4.8)$$

where  $p_{\mathcal{A}}$  is the output prediction of the action label, i.e., of dimension  $|\mathcal{A}| + 1$  for action classes and the background class, respectively. We follow the per-object regression as described in Section 4.3.2 with the equations 4.5-4.6.

**Multi-scale training and testing.** Following Ren et al. [2015a], testing is done on a fixed scale of 600 pixels. To explore the robustness to different scales, for all cases i.–vii. we also consider multi-scale testing, where we fix the scales to [480, 600, 800] pixels [Peng and Schmid, 2016]. For better comparison and comprehension of our method, for the  $ef_6$  strategy, we also consider training with multiple scales [480, 600, 800]. Our results (see paragraphs [ Multi-scale training]–[ Multi-scale testing] in Section 4.5.2) show that multi-scale testing leads to 1 – 2% better performance, while multi-scale training impacts the detection performance only by a small amount despite its long training time.

## 4.5.2 Experimental results on action adaptation

Table 4.9 shows the frame-mAP results for the five different adaptation strategies we consider as well as for the baselines. Initially, we observe that training with RGB alone outperforms training with flow alone. For the UCF-Sports and J-HMDB datasets the RGB stream achieves only 2 – 4% higher mAP than the flow stream. For UCF-101, however, there is a clear gap between the RGB and flow streams, indicating that motion is not a characteristic cue for this dataset. As far as the two baselines is concerned, we observe that the late fusion significantly outperforms the union fusion for all datasets. Additionally, late fusion achieves the highest results for the J-HMDB dataset, while it is on par with the best performance ( $e\tilde{f}_8$ ) for UCF-Sports and UCF-101.

| Description                | input |      | RoI |      | Stream Fusion       | UCF-Sports  | J-HMDB      | UCF-101     |
|----------------------------|-------|------|-----|------|---------------------|-------------|-------------|-------------|
|                            | RGB   | Flow | RGB | Flow |                     |             |             |             |
| cues alone                 | ✓     | -    | ✓   | -    | -                   | 79.3        | 48.6        | 56.4        |
|                            | -     | ✓    | -   | ✓    | -                   | 74.8        | 46.7        | 36.9        |
| Baselines                  | ✓     | ✓    | ✓   | ✓    | i. union            | 81.5        | 52.0        | 56.3        |
|                            | ✓     | ✓    | ✓   | ✓    | ii. late            | 85.1        | <b>59.6</b> | 59.7        |
| adaptive fusion strategies | ✓     | ✓    | ✓   | ✓    | iii. $ef_1$         | 79.7        | 51.8        | 58.3        |
|                            | ✓     | ✓    | ✓   | ✓    | iv. $ef_5$          | 79.5        | 48.2        | 44.6        |
|                            | ✓     | ✓    | ✓   | ✓    | v. $ef_6$           | 81.5        | 54.0        | <b>61.3</b> |
|                            | ✓     | ✓    | ✓   | ✓    | vi. $\tilde{ef}_6$  | 84.2        | 51.0        | 61.2        |
|                            | ✓     | ✓    | ✓   | ✓    | vii. $\tilde{ef}_8$ | <b>85.4</b> | 55.3        | 59.3        |

Table 4.9: *Frame-mAP* for action detection of different training scenarios on the UCF-Sports, J-HMDB and UCF-101 datasets. For more details see Section 4.5.2.

Regarding the fusion adaptation strategies, we observe the following: fusing the two streams before any fully-connected layer ( $ef_1$  and  $ef_5$ ) results in very poor performance. This indicates that the features learned in the early layers are generic and not powerful enough to represent action classes. The  $ef_6$  and  $\tilde{ef}_6$  fusions differ only in the operation for fusing the features, as the former concatenates the features from both streams, while the latter sums them. For UCF-Sports the  $\tilde{ef}_6$  outperforms the  $ef_6$ , while for J-HMDB the reverse occurs. For UCF-101, these two cases perform the same. The last strategy we examine is  $\tilde{ef}_8$ , which for UCF-Sports outperforms all other strategies. For J-HMDB,  $\tilde{ef}_8$  achieves the highest results among the five adaptation strategies, but yet its performance is still noticeably below the late fusion. For UCF-101, it achieves performance on par with late fusion, but still below the  $ef_6$  and  $\tilde{ef}_6$  fusions.

Based on the results of Table 4.9, we observe the following trends: (a) The RGB stream outperforms the flow stream, especially when the motion is not indicative for each class. (b) The late fusion is the best fusion strategy when the RGB and flow cues perform similarly (UCF-Sports and J-HMDB). (c) In cases where the motion cue is not strong (UCF-101), an adaptation method, such as  $ef_6$  or  $\tilde{ef}_6$  improves the detection performance, as the network learns when to leverage motion information, and how to jointly learn features coming from either stream.

| Description                | input |      | RoI |      | Stream Fusion       | UCF-Sports  | J-HMDB      | UCF-101     |
|----------------------------|-------|------|-----|------|---------------------|-------------|-------------|-------------|
|                            | RGB   | Flow | RGB | Flow |                     |             |             |             |
| cues alone                 | ✓     | -    | ✓   | -    | -                   | 80.8        | 50.0        | 57.5        |
|                            | -     | ✓    | -   | ✓    | -                   | 75.9        | 44.5        | 35.6        |
| Baselines                  | ✓     | ✓    | ✓   | ✓    | i. union            | 81.9        | 50.1        | 56.9        |
|                            | ✓     | ✓    | ✓   | ✓    | ii. late            | 86.4        | 55.7        | 59.6        |
| adaptive fusion strategies | ✓     | ✓    | ✓   | ✓    | iii. $ef_1$         | 82.7        | 51.8        | 58.4        |
|                            | ✓     | ✓    | ✓   | ✓    | iv. $ef_5$          | 84.2        | 51.2        | 49.9        |
|                            | ✓     | ✓    | ✓   | ✓    | v. $ef_6$           | 84.0        | 55.6        | <b>61.3</b> |
|                            | ✓     | ✓    | ✓   | ✓    | vi. $\tilde{ef}_6$  | 86.1        | 51.2        | 60.9        |
|                            | ✓     | ✓    | ✓   | ✓    | vii. $\tilde{ef}_8$ | <b>86.9</b> | <b>56.1</b> | 59.3        |

Table 4.10: *Frame-mAP for action localization of different training scenarios with multi-scale testing on the UCF-Sports, J-HMDB and UCF-101 datasets. The test scales we use are [480, 600, 800].*

**Multi-scale testing.** To explore the robustness to different scales, for all cases, we test on bounding boxes of multiple scales [480, 600, 800]. Table 4.10 reports the frame-mAP results for the multi-scale testing of all the adaptive fusion strategies. For the RGB cue alone, we observe a small increment in the performance of all datasets compared to one fixed scale (Table 4.9). In contrast, for the J-HMDB and UCF-101 datasets the performance of the flow cue marginally decreases when considering multiple scales, while for UCF-Sports it increases. Additionally, again the RGB cue alone outperforms the flow one by the same relative margin as with one scale. When considering the baselines, we observe almost no difference in the detection performance compared to single scale testing, while for the fusion strategies (iii-vii) we observe a small yet consistent increment in the detection performance, see Tables 4.9-4.10. In particular, all mAPs are increased by 1 – 2% for UCF-Sports and J-HMDB, but they remain almost constant for UCF-101. Overall, we conclude that the multi-scale testing improves the detection performance by a small margin of 1% for all datasets.

**Multi-scale training and testing.** To explore the robustness to different scales we train and test the  $ef_6$  architecture on bounding boxes of multiple scales [480, 600, 800]. Table 4.11 reports the frame-mAP results. For either testing on one or multiple scales, we observe that training with multiple scales does improves the performance only by a small, negligible margin for all datasets. In particular, UCF-Sports is the only dataset

| scales          |                 | UCF-Sports | J-HMDB | UCF-101 |
|-----------------|-----------------|------------|--------|---------|
| test            | training        |            |        |         |
| [600]           | [600]           | 81.5       | 54.0   | 61.3    |
|                 | [480, 600, 800] | 85.4       | 54.9   | 60.8    |
| [480, 600, 800] | [600]           | 84.0       | 55.6   | 61.3    |
|                 | [480, 600, 800] | 87.1       | 56.0   | 60.8    |

Table 4.11: *Frame-mAP for action localization for the  $ef_6$  fusion scenario with multi-scale training and testing on the UCF-Sports, J-HMDB and UCF-101 datasets. Both training and test scales are [480, 600, 800].*

where multi-scale training leads to an increment, whereas J-HMDB and UCF-101 are not affected by the multi-scale training. Overall, we conclude that multi-scale training barely impacts the detection performance despite its long training time.

### 4.5.3 Action adversarial adaptation

As shown in Section 4.5.1, constructing two-stream networks in an end-to-end architecture is not trivial. The RGB and flow data come from different distributions and belong to different feature spaces; learning them jointly, however, requires a common feature space. Therefore, here, we want to map the features from the RGB and flow data to a common embedding, where their features can be jointly and effectively learned. To this end, we follow recent domain adaptation techniques [Tzeng et al., 2015, 2017] to match the two distributions.

In particular, for action localization we rely on the findings of Section 4.5.1, and use the  $ef_6$  two-stream end-to-end architecture, as presented in Figure 4.7 (c). In addition, we add an extra adversarial adaptation component, that aims at learning a common embedding between RGB and flow frames. The final architecture is shown in Figure 4.8.

We design our adversarial component to take as input the *conv5* features from each stream. Then, it feeds them to a series of three fully-connected layers. This component serves as a common embedding for the features of the data of the two domains. The features from both domains are mapped to intermediate feature spaces, and we aim at minimizing the distance between these two embeddings. To this end, we introduce the domain confusion loss that aims at domain invariance, i.e., not being able to distinguish from which domain the data come. Optimizing for this loss alone though, would result



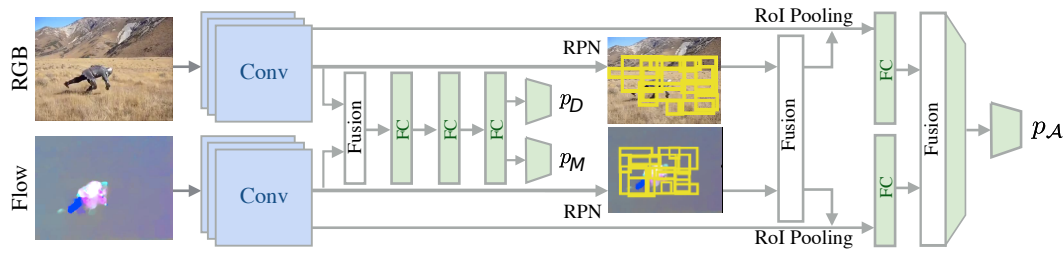


Figure 4.8: Illustration of the adversarial fusion network. The two streams are fused in the  $fc6$  and at the  $conv5$ , where two losses are computed:  $p_D$  for the domain discriminator and  $p_M$  for the domain confusion loss.

in assigning all data to one domain. To avoid this, we include in the overall objective an adversarial discriminator that tries to distinguish from which domain the data come. These two losses may seem contradictory as they aim at opposite goals. They are however essential for good learning, as the latter (adversarial discriminator) keeps trying to classify correctly the frames, while the former (domain confusion) keeps trying to create such a robust feature embedding so as to confuse the adversarial discriminator.

In the following, we describe the losses and training procedure with more details.

**Losses.** Initially, we consider two domains: (a) source  $S$  comprising data  $x$  drawn from a distribution  $p_S$  and (b) target  $T$  comprising data  $y$  drawn from a distribution  $p_T$ . For instance, we consider the source data to be the RGB frames, and the target data to be the flow frames. Our goal is combine the features from the two distributions.

First, we introduce an adversarial discriminator  $D$  that aims to distinguish between a mapped source sample  $M_S(x)$  and a mapped target samples  $M_T(y)$ . For brevity, we call  $D$  the domain discriminator, that is optimized according to the standard loss [Liu and Tuzel, 2016]:

$$L_{advD}(S, T, M_S, M_T) = -\mathbb{E}_{x \sim p_S} [\log D(M_S(x))] - \mathbb{E}_{y \sim p_T} [\log(1 - D(M_T(y)))] \quad . \quad (4.9)$$

In practice, having this loss alone in the objective function would result in totally distinguishable feature representations, which is the opposite of a common feature embedding. To this end, we map the source and target data to two embeddings  $M_S$  and  $M_T$ , respectively, and we aim at minimizing their distance. Following Tzeng et al. [2015, 2017], we introduce a domain confusion loss:

$$L_{advM}(S, T, D) = -\mathbb{E}_{y \sim p_T} [\log D(M_T(y))] \quad . \quad (4.10)$$

This domain confusion loss ensures that the features from one domain are indistinguishable from the features of the other domain, i.e., domain invariance. Note that this loss enforces assigning the target data as source data. Having this loss alone in the objective function would result assigning the source label to all data, regardless the domain they come from. Using both losses results in the following adversarial objective:

$$L_O = L_{\text{adv}D}(S, T, M_S, M_T) + L_{\text{adv}M}(S, T, D) \quad . \quad (4.11)$$

Apart from the adversarial loss, our network (Figure 4.8) comprises also the detection loss, i.e., the classification and regression losses as well as the RPN losses for the RGB and flow streams. Finally, the overall objective loss of the network is:

$$\mathcal{L} = L_O + \mathcal{L}_{\text{RPN}_R} + \mathcal{L}_{\text{RPN}_F} + \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{reg}} \quad , \quad (4.12)$$

where  $\mathcal{L}_{\text{cls}}$  is defined in Equation 4.8, and  $\mathcal{L}_{\text{reg}}$  in Equation 4.5.

**Training.** The network of Figure 4.8 takes as input RGB and flow frames and has two losses, the overall adversarial loss and the detection loss. We perform two independent trainings, one considering as source data the RGB stream, and another one where we consider as source data the flow stream. Then we use late score fusion to score the final results coming from each training, see Section 4.5.1 for more details on the late fusion.

The training of the source stream differs from the one of the target stream in the back-propagation of the losses. In particular, the gradients as derived by the adversarial loss are back-propagated only to the target stream. This is because the adversarial component learns an embedding of the target data that makes their representation more similar to the source ones. In contrast, the gradients of the detection loss are back-propagated to both streams. In summary, the source stream operates under the detection loss, while only the target stream operates under the action adversarial loss for action detection of Equation 4.12.

#### 4.5.4 Experimental results on action adversarial adaptation

In this section, we evaluate our action adversarial adaptation approach for action localization, see Figure 4.8. We perform experiments on the UCF-Sports, J-HMDB, and UCF-101 datasets, as described in Section 2.4.4.

| training scales | source      | target | UCF-Sports  | J-HMDB      | UCF-101     |
|-----------------|-------------|--------|-------------|-------------|-------------|
| [600]           | RGB         | flow   | 82.3        | 52.3        | 60.1        |
|                 | flow        | RGB    | 81.9        | 51.2        | 60.0        |
|                 | late fusion |        | 82.7        | <b>53.6</b> | 60.3        |
| [480, 600, 800] | RGB         | flow   | 84.2        | 52.5        | 61.3        |
|                 | flow        | RGB    | 83.0        | 52.2        | 61.0        |
|                 | late fusion |        | <b>83.8</b> | 53.4        | <b>61.5</b> |

Table 4.12: *Frame-mAP* for action localization for the  $\tilde{e}f_6$  adversarial network with one and multiple training scales on the UCF-Sports, J-HMDB and UCF-101 datasets.

We train the network twice, one considering the RGB data as source domain and the flow data as target and vice versa. Table 4.12 reports the frame-mAP results. When the RGB stream is used as source to adapt the flow stream, we observe a big increment in the performance compared to using just the flow stream. But this increment is still inferior to some of the fusion methods presented in Section 4.5.1 (see also Tables 4.9–4.11). In the reverse scenario, i.e., when the flow data are the source domain targeting at the RGB frames, we observe a bit lower detection performance than when using RGB as source data. This can be explained by considering the lower performance the flow stream has compared to the RGB one. In particular, we adapt the RGB features to be more similar to the flow ones, that originally perform worse than the RGB.

Surprisingly, the overall result is in the same magnitude with the ones from the fusion strategies explored in Section 4.5.1. This reveals that the adversarial adaptation technique we examine is not sufficient to improve the action localization alone. In particular, for UCF-101 all the methods we explore perform almost equally ( $\pm 1\%$ ), suggesting that stronger methodology is needed. For instance, a better training scheme that includes alternate training could improve the performance. Independent of the method, another alternative would be to exploit the best possible model with the current architecture using more training data [Carreira and Zisserman, 2017]. Finally, we observe that the late fusion and the multi-scale training improves the performance only by a small margin.

## 4.6 Conclusions

Most state-of-the-art works for video detection aim at localizing either objects *or* actions. Instead, we jointly detect objects and actions in uncontrolled video scenes, e.g. *car rolling*. To this end, we propose an end-to-end network built upon Faster R-CNN [Ren et al., 2015a]. The key point is that our network operates under a multitask objective. We show that this joint training via the proposed multitask objective: (a) outperforms training alone with objects or with actions, as the network is better able to generalize, less prone to overfit and benefits from sharing statistical strength between classes, (b) performs as well as other variants while requiring fewer parameters and (c) allows zero-shot learning of actions performed by an object, for which no action labels are present at training time. Our network can also be applied to different tasks including semantic segmentation and visual relationships between objects. We also apply our end-to-end network to the action localization task, where we explore possible (adversarial) adaptation stream-fusion techniques. We observe that adapting the RGB and flow streams improves the detection performance. This adaptation though cannot solve the action localization problem on its own; we need to exploit other dimensions of videos, such as the temporal one, see Chapter 5. Appendix D shows additional results.



# Chapter 5

## Action Tubelet Detector for Spatio-Temporal Action Localization

### Contents

---

|   |            |
|---|------------|
| <b>5.1 Overview</b>                                 | <b>120</b> |
| <b>5.2 Related work</b>                             | <b>122</b> |
| <b>5.3 Action tubelet detector</b>                  | <b>125</b> |
| 5.3.1 ACT-detector                                  | 125        |
| 5.3.2 Two stream ACT-detector                       | 129        |
| 5.3.3 From action tubelets to spatio-temporal tubes | 129        |
| <b>5.4 Experimental results</b>                     | <b>131</b> |
| 5.4.1 Datasets and metrics                          | 131        |
| 5.4.2 Implementation details                        | 132        |
| 5.4.3 Validation of anchor cuboids                  | 133        |
| 5.4.4 Tubelet modality                              | 135        |
| 5.4.5 Tubelet length                                | 135        |
| 5.4.6 Error breakdown analysis                      | 138        |
| 5.4.7 Handling moving actors                        | 139        |
| 5.4.8 Comparison to other linking methods           | 140        |
| 5.4.9 Comparison to the state of the art            | 141        |
| <b>5.5 Conclusions</b>                              | <b>144</b> |

---

Action localization focuses both on classifying the actions present in a video and on localizing them in space and time. CNNs have proven to be well adapted for action localization, as they provide robust representations of video frames. Indeed, most state-of-the-art action localization approaches [Gkioxari and Malik, 2015; Peng and Schmid, 2016; Saha et al., 2016; Singh et al., 2017; Weinzaepfel et al., 2015] are based on CNN object detectors [Liu et al., 2016a; Ren et al., 2015a] that detect human actions at the frame level. Then, they either link frame-level detections, or track them over time, to create spatio-temporal tubes. Similarly, in Chapters 3-4, we focus on detecting moving objects and actions *only* for single frames.

In this chapter, however, we leverage the temporal continuity of videos instead of operating at the frame or pairs of frames level. We propose the **ACTion Tubelet** detector (ACT-detector) that takes as input a sequence of frames and outputs *tubelets*, i.e. sequences of bounding boxes with associated scores. This work is published in [Kalogeiton et al., 2017a] and the code is available at <http://thoth.inrialpes.fr/src/ACTdetector>.

**Outline.** This chapter is organized as follows. We first present an overview of our method (Section 5.1), and then we review related works in Section 5.2. Next, Section 5.3 describes our proposed ACT-detector. We present extensive experimental results on object detection alone and on object-action detection in Section 5.4. We finally draw conclusions in Section 5.5. In Appendix E we report per-class results for the experiments of this chapter.

## 5.1 Overview

Albeit their remarkable results [Peng and Schmid, 2016; Saha et al., 2016], state-of-the-art action localization methods do not truly exploit the temporal continuity of videos, as they treat the video frames as a set of independent images on which a detector is applied independently. Nevertheless, processing frames individually is not optimal, as distinguishing actions from a single frame can be ambiguous, e.g. *person sitting down* or *standing up* (Figure 5.1).

In this chapter, we propose to surpass this limitation and treat a video as a sequence of frames. State-of-the-art object detectors for images, such as Faster R-CNN Ren et al. [2015a] and Single Shot MultiBox Detector (SSD) Liu et al. [2016a] (Section 2.2), proceed by classifying and regressing a set of anchor boxes to the true bounding box



Figure 5.1: *Understanding an action from a single frame can be ambiguous, e.g. sitting down or standing up; the action becomes clear when looking at a sequence of frames.*

of the object. In this chapter, we introduce a spatio-temporal tubelet extension of this design. Our Action Tubelet detector (ACT-detector) takes as input a short sequence of a fixed number of frames and outputs *tubelets*, i.e., sequences of bounding boxes over time (Figure 5.2). Our method considers densely sampled anchors of cuboid shape with various sizes and aspect ratios. At test time, we generate for each anchor cuboid a score for a given action and regressed coordinates transforming it into a tubelet. Importantly, the score and regression are based on convolutional feature maps from all frames in the sequence. While the anchor cuboids have fixed spatial extent across time, the tubelets change size, location and aspect ratio over time, following the actors. Here we build upon the state-of-the-art Single Shot MultiBox Detector (SSD) framework, but the proposed tubelet extension is applicable to other detectors based on anchor boxes, such as Faster R-CNN.

Our experiments show that taking as input a sequence of frames improves: (a) action scoring, because the ambiguity between different actions reduces and (b) localization accuracy, because frames in a cuboid are regressed jointly and hence, they share information about the location of the actor in neighboring frames, see Figure 5.1. Our ACT-detector obtains state-of-the-art frame-mAP and video-mAP performance on the J-HMDB [Jhuang et al. \[2013a\]](#) and UCF-101 [Soomro et al. \[2012\]](#) action localization datasets, in particular at high overlap thresholds.

In summary, we make the following contributions:



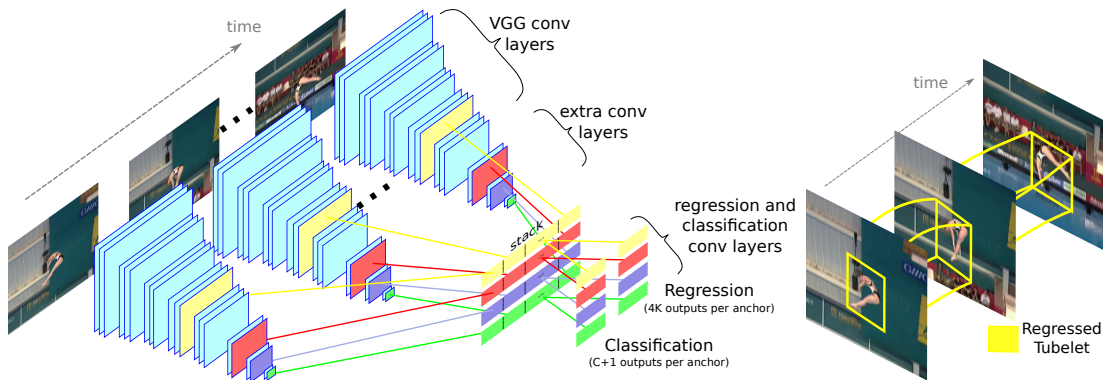


Figure 5.2: Overview of our ACT-detector. Given a sequence of frames, we extract convolutional features with weights shared between frames. We stack the features from subsequent frames to predict scores and regress coordinates for the anchor cuboids (middle figure, blue color). Depending on the size of the anchors, the features come from different convolutional layers (left figure, color coded: yellow, red, purple, green). As output, we obtain tubelets (right figure, yellow color).

- We introduce the ACT-detector, an action tubelet detector that proceeds by scoring and regressing anchor cuboids.
- We demonstrate that anchor cuboids can handle moving actors for sequences up to around 10 frames.
- We provide an extensive analysis demonstrating the clear benefit of leveraging sequences of frames instead of operating at the frame level.

The code of our ACT-detector is available at <http://thoth.inrialpes.fr/src/ACTdetector>.

## 5.2 Related work

Almost all recent work [Peng and Schmid, 2016; Saha et al., 2016; Singh et al., 2017; Weinzaepfel et al., 2015] for spatio-temporal action localization builds on state-of-the-art CNN object detectors [Liu et al., 2016a; Ren et al., 2015a]. In the following, we first review some recent CNN object detectors and then examine the most relevant state-of-the-art action localization approaches. For more details, refer to Sections 2.2-2.4.

**Object detection with CNNs.** Recent state-of-the-art object detectors [Girshick et al., 2014a; Liu et al., 2016a; Redmon et al., 2016; Ren et al., 2015a] are based on CNNs. R-CNN of Girshick et al. [2014a] casts the object detection task as a region-proposal

classification problem. Faster R-CNN of [Ren et al. \[2015a\]](#) extends this approach by generating bounding box proposals with a fully-convolutional Region Proposal Network (RPN). RPN considers a set of densely sampled anchor boxes, that are scored and regressed. Moreover, it shares convolutional features with proposal classification and regression branches. These branches operate on fixed-size dimension features obtained using a Region-of-Interest (RoI) pooling layer. In a similar spirit, YOLO [[Redmon et al., 2016](#)] and SSD [[Liu et al., 2016a](#)] (Section 2.2.5) also use a set of anchor boxes, which are directly classified and regressed without a RoI pooling layer. In YOLO, all scores and regressions are computed from the last convolutional feature maps, whereas SSD adapts the features to the size of the boxes, whereas SSD adapts the features to the size of the boxes. Features for predicting small-sized boxes come from early layers, and features for big boxes come from the latter layers, with larger receptive fields. All these object detectors rely on a set of anchor boxes. In our work, we extend them to anchor cuboids leading to significant improvement for action localization in videos.

**Action localization.** Initial approaches for spatio-temporal action localization in videos were extensions of the sliding window scheme [[Cao et al., 2010](#); [Laptev and Pérez, 2007](#)], requiring strong assumptions such as a cuboid shape, i.e., a fixed spatial extent of the actor across frames. Other methods extend object proposals to videos. Hundreds of action proposals are extracted per video given low-level cues, such as super-voxels [[Jain et al., 2014](#); [Oneata et al., 2014a](#)] or dense trajectories [[Chen and Corso, 2015](#); [Gemert et al., 2015](#); [Marian Puscas et al., 2015](#)]. Then, they cast action localization as a proposal classification problem, see Section 2.4.2 for more details.

More recently, some approaches [[Li et al., 2016d](#); [Wang et al., 2016a](#); [Yu and Yuan, 2015](#)] rely on an actionness measure [[Chen et al., 2014b](#)], i.e. a pixel-wise probability of containing any action. To estimate actionness, they use low-level cues such as optical flow [[Yu and Yuan, 2015](#)], CNNs with a two-stream fully-convolutional architecture [[Wang et al., 2016a](#)] or recurrent neural networks [[Li et al., 2016d](#)]. They extract action tubes either by thresholding [Li et al. \[2016d\]](#) the actionness score, or by using a maximum set coverage formulation [[Yu and Yuan, 2015](#)]. This, however, outputs only a rough localization of the action as this localization is based on noisy pixel-level maps.

Most recent approaches rely on object detectors trained to discriminate human action classes at the frame level. [Gkioxari and Malik \[2015\]](#) extend the R-CNN framework to a two-stream variant [[Simonyan and Zisserman, 2014](#)], processing RGB and

flow data separately. Then, the resulting per-frame detections are linked using dynamic programming with a cost function based on detection scores of the boxes and overlap between detections of consecutive frames. Weinzaepfel et al. [2015] replace the linking algorithm by a tracking-by-detection method.

More recently, two-stream Faster R-CNN was introduced by [Peng and Schmid, 2016; Saha et al., 2016]. Saha et al. [2016] fuse the scores of both streams based on overlap between the appearance and the motion RPNs. Peng and Schmid [2016] combine proposals extracted for the two streams and then classify and regress them with fused RGB and multi-frame optical flow features. They also use multiple regions inside each action proposal and then link the detections across a video based on spatial overlap and classification score.

The two-stream architecture is also used by Singh et al. [2017]. They perform action localization in real-time using (a) the efficient SSD detector, (b) a fast method [Kroeger et al., 2016] to estimate the optical flow for the motion stream, and (c) an online linking algorithm. All these approaches rely on detections *at the frame level*. In contrast, we build our ACT-detector by taking as input sequences of frames and demonstrate improved action scores and location accuracy over frame-level detections.

**Linking and tracking methods for action localization.** Action localization methods detect per-frame actions, and then link or track these detections over time. Given the per-frame detections for each class, Weinzaepfel et al. [2015] select the highest scored ones and track them throughout the video. Their tracking-by-detection method leverages an instance-level CNN detector and a class-level classifier. They perform the tracking multiple times for each action, starting from the detection with the highest score that do not overlap with previous computed tracks. In the end, they perform temporal detection using a multiscale sliding window approach: for each class, they slide a temporal window over the tracks and compute its score. Each score is computed from the CNN features, the histograms of gradient, and motion of each spatio-temporal cell and a duration prior. The temporal windows they consider have different length. Finally, for each track they select the window with the highest score.

More recent action localization methods perform linking of detections instead of tracking, as the modern per-frame detectors result in powerful scoring and localization. The method of Peng and Schmid [2016] links detections over time based on their overlap and score. At the first frame of the video, they consider its detections after NMS for each class. For each detection in the current frame, they compute a metric with all

the detections of the next frame. Their metric is the sum over the scores of the two boxes and their spatial IoU (the box from the current frame and each one of the next frame). Then, they link the detections that maximize this metric over time. In order to determine the temporal extent of an action detection within a video track, they apply a sliding window approach with multiple temporal scales and strides as [Weinzaepfel et al., 2015]. The method of Singh et al. [2017] is a greedy method that build incrementally (frame by frame) tubes for each action class. At the first frame of the video, they initialize  $N$  tubes for each class using  $N$  boxes. Then, for each class and for each tube, they add one box at a time, i.e. for each new frame. They select the box based on *IoU* matching, i.e. they keep the box with the highest score that verifies  $IoU > \text{threshold}$ . At each time step, they sort the existing tubes. They terminate the tube generation, if after  $m$  frames there are no matches. Finally, each tube is temporally trimmed by performing a binary labelling using an online Viterbi algorithm [Viterbi, 1967].

## 5.3 Action tubelet detector

We introduce the **ACTion Tubelet** detector (ACT-detector), an action tubelet approach for action localization in videos. The ACT-detector takes as input a sequence of  $K$  frames  $f_1, \dots, f_K$  and outputs a list of spatio-temporal detections, each one being a *tubelet*, i.e., a sequence of bounding boxes, with one confidence score per action class. The idea of such an extension to videos could be applied on top of various state-of-the-art object detectors. Here, we apply our method on top of SSD, as it has lower runtime than other detectors, which makes it suitable for large video datasets. In this section, we first describe our proposed ACT-detector (Section ??), and then our full framework for video detection (Section 5.3.2). Finally, Section 5.3.3 describes our method for constructing action tubes.

### 5.3.1 ACT-detector

In this chapter, we claim that action localization benefits from predicting tubelets taking as input a sequence of frames instead of operating at the frame level. Indeed, the appearance and even the motion may be ambiguous for a single frame. Considering more frames for predicting the scores reduces this ambiguity (Figure 5.1). Moreover, this allows to perform regression jointly over consecutive frames, instead of doing it independently for each of them. Our ACT-detector builds upon SSD, see Figure 5.2 for

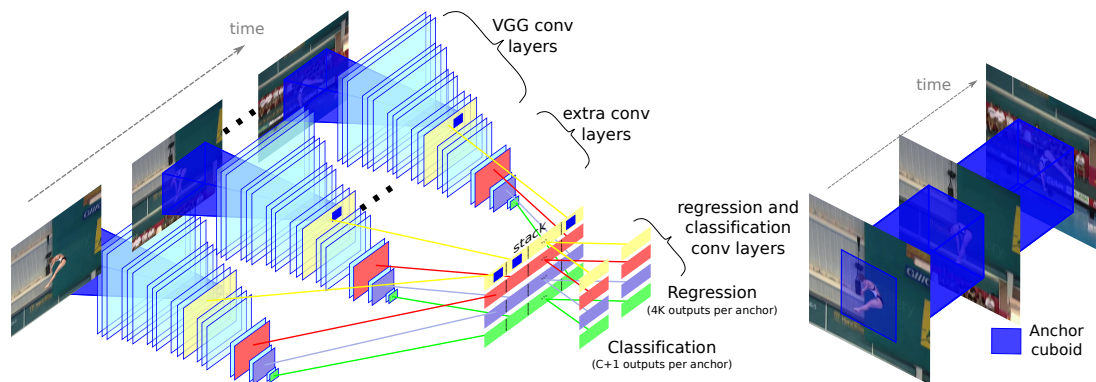


Figure 5.3: *Our ACT-detector: construction of the anchor cuboids from different convolutional layers, such as the blue color in the figure. (right) The anchor cuboid over time.*

an overview of the approach. In the following we review the SSD detector in details and then present our ACT-detector.

**SSD detector.** The SSD detector (Single Shot MultiBox Detector) [Liu et al. \[2016a\]](#) performs object detection by considering a set of anchor boxes at different positions, scales and aspect ratios. Each of them is (a) scored for each object class and for a background class, and (b) regressed to better fit the object extent. SSD uses a fully convolutional architecture, without any object proposal step, enabling fast computation. The classification and regression are performed using different convolutional layers depending on the scale of the anchor box. Note that the receptive field of a neuron used to predict the classification scores and the regression of a given anchor box remains significantly larger than the box, see Section 2.2 for more details about anchor boxes. For more details about SSD, refer to Section 2.2.5.

**Our ACT-detector** Given a sequence of  $K$  frames, the ACT-detector computes convolutional features for each one. The *weights* of these convolutional features are shared among all input frames, thus enabling joint learning of the features. We extend the anchor boxes of SSD to anchor cuboids by assuming that the spatial extent is fixed over time along the  $K$  frames.

We then stack the corresponding convolutional features from each of  $K$  frames (Figure 5.3). The stacked features are the input of two convolutional layers, one for scoring action classes and one for regressing the anchor cuboids. For instance, when considering an anchor cuboid for which the prediction is based on the ‘red’ feature

maps of Figures 5.2-5.3, the classification and regression are performed with convolutional layers that take as input the ‘red’ stacked feature maps from the  $K$  frames. The classification layer outputs for each anchor cuboid  $C + 1$  scores: one per action class plus one for the background. This means that the tubelet classification is done based on the sequence of frames. The regression outputs  $4 \times K$  coordinates (4 for each of the  $K$  frames) for each anchor cuboid. Note that although all boxes in a tubelet are regressed jointly, they result in a different regression for each frame.

The initial anchor cuboids have a fixed spatial extent over time, see Figure 5.4:top. We compute the anchor cuboids by considering the same set of anchor boxes from SSD and extending them over time. In Section 5.4.3 we show experimentally that such anchor cuboids can handle moving actors for short sequences of frames. Note that the receptive field of the neurons used to score and regress an anchor cuboid is larger than its spatial extent. This allows us to base the prediction also on the context around the cuboid, i.e., with knowledge for actors that may move outside the cuboid. Moreover, the regression significantly deforms the cuboid shape.

Even though anchor cuboids have fixed spatial extent, the tubelets obtained after regressing the  $4 \times K$  coordinates do not. We display two examples in Figure 5.4 with the anchor cuboid (cyan boxes) and the resulting regressed tubelet (yellow boxes). Note how the regression outputs an accurate localization despite the change in aspect ratio of the action boxes across time.

**Training loss.** For training, we consider only sequences of frames in which all frames contain the ground-truth action. As we want to learn action tubes, all positive and negative training data come from sequences in which actions occur. We exclude sequences in which the action starts or ends.

Let  $\mathcal{A}$  be the set of anchor cuboids. We denote by  $\mathcal{P}$  the set of anchor cuboids for which at least one ground-truth tubelet has an overlap over 0.5, and by  $\mathcal{N}$  the complementary set. Overlap between tubelets is measured by averaging the Intersection over Union (IoU) between boxes over  $K$  frames. Each anchor cuboid from  $\mathcal{P}$  is assigned to ground-truth boxes with IoU over 0.5. More precisely, let  $x_{ij}^y \in \{0, 1\}$  be the binary variable whose value is 1 if and only if the anchor cuboid  $a_i$  is assigned to the ground-truth tubelet  $g_j$  of label  $y$ .

The training loss  $\mathcal{L}$  is defined as:

$$\mathcal{L} = \frac{1}{N} (\mathcal{L}_{\text{conf}} + \mathcal{L}_{\text{reg}}) , \quad (5.1)$$

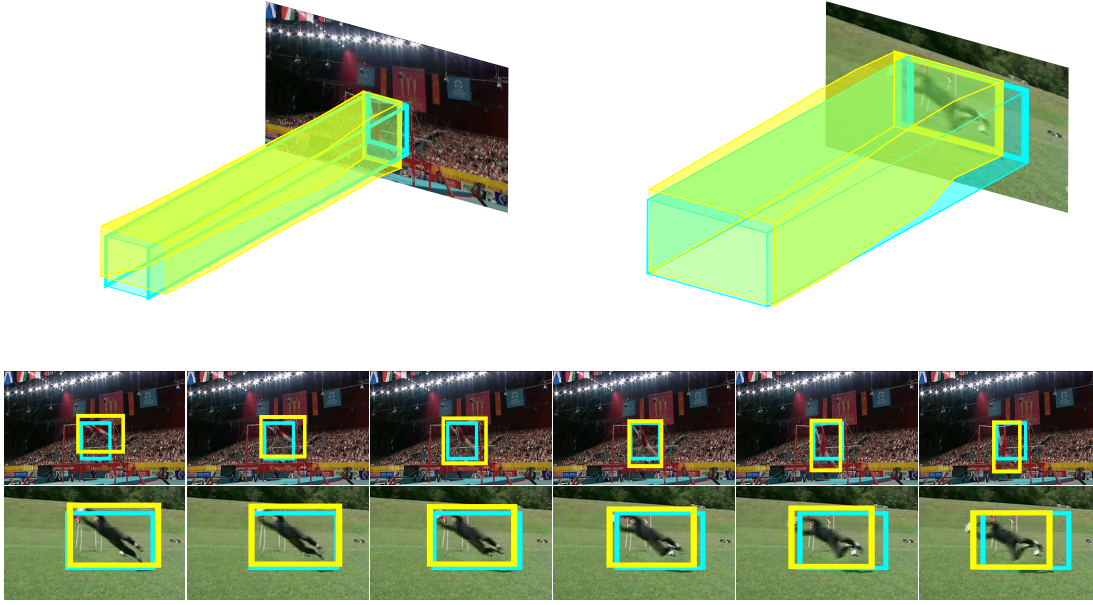


Figure 5.4: Example of regressed tubelet (yellow) from a given cuboid (cyan) in our proposed ACT-detector. Note the accurate localization of the tubelet, despite the fact that the aspect ratio of the cuboid is changing heavily across time.

with  $N = \sum_{i,j,y} x_{ij}^y$  the number of positive assignments and  $\mathcal{L}_{\text{conf}}$  (resp.  $\mathcal{L}_{\text{reg}}$ ) the confidence (resp. regression) loss as defined below.

The confidence loss is defined using a softmax loss. Let  $\hat{c}_i^y$  be the predicted confidence score (after softmax) of an anchor  $a_i$  for class  $y$ . The confidence loss is:

$$\mathcal{L}_{\text{conf}} = - \sum_{i \in \mathcal{P}} x_{ij}^y \log(\hat{c}_i^y) - \sum_{i \in \mathcal{N}} \log(\hat{c}_i^0) . \quad (5.2)$$

The regression loss is defined using a Smooth-L1 loss between the predicted regression and the ground-truth target. We regress an offset for the center  $(x, y)$  of each box in the tubelet, as well as for the width  $w$  and the height  $h$ . The regression loss is averaged over  $K$  frames.

More precisely, let  $\hat{r}_i^{xk}$  be the predicted regression for the  $x$  coordinate of anchor  $a_i$  at frame  $f_k$  and let  $g_j$  be the ground-truth target. The regression loss is defined as:

$$\begin{aligned} \mathcal{L}_{\text{reg}} &= \frac{1}{K} \sum_{i \in \mathcal{P}} \sum_{c \in \{x, y, w, h\}} x_{ij}^y \sum_{k=1}^K \text{SmoothL1}(\hat{r}_i^{ck} - g_{ij}^{ck}) , \\ \text{with } g_{ij}^{xk} &= \frac{g_j^{xk} - a_i^{xk}}{a_i^{wk}} & g_{ij}^{yk} &= \frac{g_j^{yk} - a_i^{yk}}{a_i^{hk}} , \\ g_{ij}^{wk} &= \log\left(\frac{g_j^{wk}}{a_i^{wk}}\right) & g_{ij}^{hk} &= \log\left(\frac{g_j^{hk}}{a_i^{hk}}\right) . \end{aligned} \quad (5.3)$$



### 5.3.2 Two stream ACT-detector

Following standard practice for action localization [Peng and Schmid, 2016; Saha et al., 2016; Weinzaepfel et al., 2015], we use a two-stream detector. We train an appearance detector, for which the input is a sequence of  $K$  consecutive RGB frames, and a motion detector, which takes as input the flow images computed with [Brox et al., 2004] and obtained following [Gkioxari and Malik, 2015].

Each stream outputs a set of regressed tubelets with confidence scores, originating from the same set of anchor cuboids. Therefore, the final outputs are two sets of tubelet detections, each coming from one stream.

For combining the two streams at test time we compare two approaches: union fusion and late fusion. For the union fusion as presented by [Singh et al., 2017], we consider the *set union* of the outputs from both streams: the tubelets from the RGB stream with their associated scores and the tubelets from the flow stream with their scores. That means that the number of tubelets is doubled (RGB and flow outputs). For the late fusion [?], for each anchor cuboid we average the scores from both streams, as the set of anchors is the same for both streams (we use the SSD pre-defined set of anchor boxes extended to anchor cuboids). We keep the regressed tubelet from the RGB stream, as appearance is more relevant for regressing boxes, in particular for actions with limited motion. Our experiments show that the late fusion outperforms the union fusion (Section 5.4.4).

### 5.3.3 From action tubelets to spatio-temporal tubes

The extracted tubelets describe the short parts of the videos. For long-term dependencies, we need spatio-temporal tubes (action tubes), i.e., spatio-temporal bounding boxes that span to the whole temporal extend of the action in the video. For constructing action tubes, we build upon the frame linking algorithm of [Singh et al., 2017], as it is robust to missed detections and can generate tubes spanning different temporal extents of the video (Section 5.2). We extend their algorithm from frame linking to *tubelet linking* and propose a *temporal smoothing* to build action tubes from the linked tubelets. The method is online and proceeds by iteratively adding tubelets to a set of links while processing the frames. In the following,  $t$  is a tubelet and  $L$  a link, i.e., a sequence of tubelets. Figure 5.5 shows an illustration of the linking tubelets process we follow.



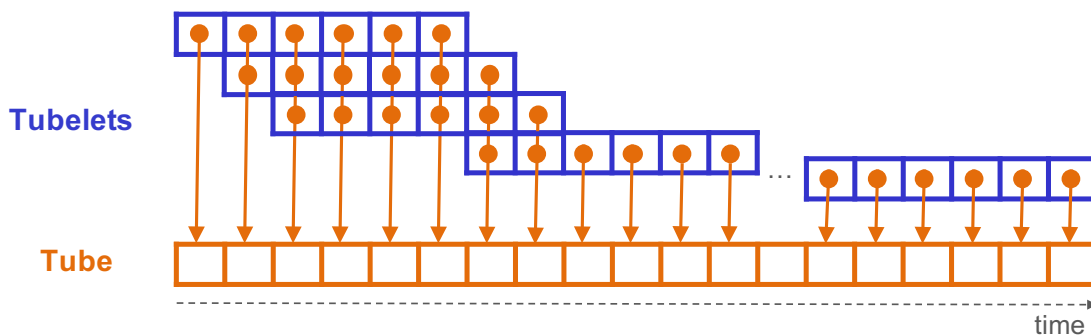


Figure 5.5: Toy example of constructing tubes (orange) from tubelets (blue). Tubelets overlap over  $K$  frames. At each frame we select the tubelet with the highest score and high overlap with the last tubelet of the link. The score and bounding boxes of a tube are the average of the scores and bounding box coordinates of its tubelets.

**Input tubelets.** Given a video, we extract tubelets for each sequence of  $K$  frames. This means that consecutive tubelets overlap by  $K-1$  frames. The computation of overlapping tubelets can be performed at an extremely low cost as the weights of the convolutional features are shared. We compute the convolutional features for each frame only once. For each sequence of frames, only the last layers that predict scores and regressions, given the stacked convolutional features (Figure 5.2), remain to be computed. For linking, we keep only the  $N=10$  highest scored tubelets for each class after non-maximum suppression (NMS) at a threshold 0.3 in each sequence of frames.

**Overlap between a link and a tubelet.** Our linking algorithm relies on an overlap measure  $ov(L, t)$  between a link  $L$  and a tubelet  $t$  that temporally overlaps with the end of the link. We define the overlap between  $L$  and  $t$  as the overlap between the last tubelet of the link  $L$  and  $t$ . The overlap between two tubelets is defined as the average IoU between their boxes over overlapping frames.

**Initialization.** In the first frame, a new link is started for each of the  $N$  tubelets. At a given frame, new links start from tubelets that are not associated to any existing link.

**Linking tubelets.** Given a new frame  $f$ , we extend one by one in descending order of scores each of the existing links with one of the  $N$  tubelet candidates starting at this frame. The score of a link is defined as the average score of its tubelets. To extend a link  $L$ , we pick the tubelet candidate  $t$  that meets the following criteria: (i) is not already selected by another link, (ii) has the highest score, and (iii) verifies  $ov(L, t) \geq \tau$ , with  $\tau$  a given threshold. In our experiments we use  $\tau=0.2$ .

**Termination.** Links stop when these criteria are not met for more than  $K - 1$  consecutive frames.

**Temporal smoothing: from tubelet links to action tubes.** For each link  $L$ , we build an action tube, i.e., a sequence of bounding boxes. The score of a tube is set to the score of the link, i.e., the average score over the tubelets in the link. To set the bounding boxes, note that we have multiple box candidates per frame as the tubelets are overlapping. One can simply use the box of the highest scored tubelet. Instead, we propose a temporal smoothing strategy. For each frame, we average the box coordinates of tubelets that pass through that frame. This allows us to build smooth tubes.

**Temporal detection.** The initialization and termination steps result in tubes spanning different temporal extents of the video. Each tube determines, thus, the start and end in time of the action it covers. No further processing is required for temporal localization.

## 5.4 Experimental results

In this section we study the effectiveness of our ACT-detector for action localization in videos. After presenting the datasets (Section 5.4.1) and the implementation details (Section 5.4.2) used in our experiments, we provide an analysis of our proposed tubelet detector. First, we validate our anchor cuboids (Section 5.4.3) and evaluate input modalities (RGB and flow) and their fusion (Section 5.4.4). Next, we examine the impact of the length  $K$  of the sequence of frames (Section 5.4.5) and present an error analysis (Section 5.4.6). Then, we show that our tubelets can handle moving actors (Section 5.4.7). These experiments demonstrate the benefit of tubelets in terms of classification and localization accuracy. Next, we compare our linking method to existing linking methods (Section 5.4.8), and finally we compare our approach to the state of the art (Section 5.4.9). Note that in this section, we only report results averaged over all classes; for per-class results, refer to Appendix E.

### 5.4.1 Datasets and metrics

**Datasets.** In our experiments, we use the three action localization datasets described in Section 2.4.4: UCF-Sports, J-HMDB, and UCF-101. They contain from 150 to 3207 videos with sports and everyday action types. For UCF-Sports we follow the train/test splits from [Lan et al. \[2011\]](#). For J-HMDB we report results averaged on the

three splits defined in [Jhuang et al., 2013a], unless stated otherwise. For UCF-101 we report results only on the first split.

**Metrics.** We use metrics at both frame and video level. Frame-level metrics allow us to compare the quality of the detections independently of the linking strategy. Metrics at the video level are the same as the ones at the frame level, replacing the Intersection-over-Union (IoU) between boxes by a spatio-temporal overlap between tubes, i.e., an average across time of the per-frame IoU [Saha et al., 2016; Weinzaepfel et al., 2015]. To measure our performance at the frame level, we take into account the boxes originating from all tubelets that pass through the frame with their individual scores and perform NMS. In all cases, we only keep the detections with score above 0.01.

We report *frame* and *video mean Average Precision (mAP)*. A detection is correct if its IoU with a ground-truth box or tube is greater than 0.5 and its action label is correctly predicted [Everingham et al., 2007]. For each class, we compute the average precision (AP) and report the average over all classes.

To evaluate the quality of the detections in terms of localization accuracy, we also report *MABO* (Mean Average Best Overlap) [Uijlings et al., 2013]. We compute the IoU between each ground-truth box (or tube) and our detections. For each ground-truth box (or tube), we keep the overlap of the best overlapping detection (BO) and, for each class, we average over all boxes (or tubes) (ABO). The mean is computed over all classes (MABO).

To evaluate the quality of the detections in terms of scoring, we also measure *classification accuracy*. In each frame, assuming that the ground-truth localization is known, we compute class scores for each ground-truth box by averaging the scores from the detected boxes or tubelets (after regression) whose overlap with the ground-truth box of this frame is greater than 0.7. We then assign the class having the highest score to each of these boxes and measure the ratio of boxes that are correctly classified.

### 5.4.2 Implementation details

We use VGGNet [Simonyan and Zisserman, 2015] as a base architecture with images of size  $300 \times 300$ . We use ImageNet pre-training for both appearance and motion streams [Peng and Schmid, 2016; Wang et al., 2016b]. We use the same hard negative mining strategy as SSD [Liu et al., 2016a], i.e., to avoid an unbalanced factor between positive and negative training samples, only the hardest negative up to a ratio of 3 negatives for 1 positive are kept in the loss. Following SSD, we perform data augmentation

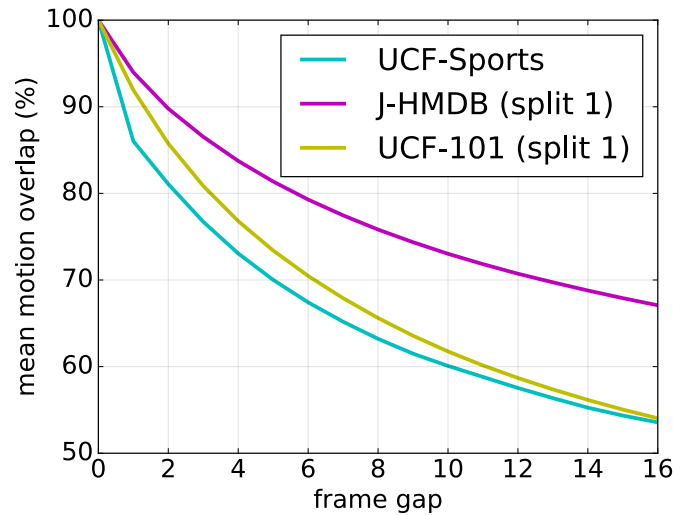


Figure 5.6: *Motion overlap: Mean motion overlap between a box in a ground-truth tube and its box  $n$  frames later for varying  $n$ .*

applied to the whole sequence of frames: photometric transformation, rescaling and cropping. Given that we have  $K$  parallel streams, the gradient of the shared convolutional layers is the sum over the  $K$  streams. We find that dividing the learning rate of the shared convolutional layers by  $K$  helps convergence, as it prevents large gradients.

### 5.4.3 Validation of anchor cuboids

Cuboids enhance the detection accuracy as they are able to capture short-term dependencies between frames. This section demonstrates that an anchor cuboid *can* handle moving actions. We first measure how much the actors move in the training sets of the three action localization datasets by computing the *mean motion overlap*. For each box in a ground-truth tube, we measure its *motion overlap*: the overlap between this box and the ground-truth box  $n$  frames later for varying  $n$ . For each class, we compute the average motion overlap over all frames and we report the mean over all classes in Figure 5.6.

We observe that the motion overlap reduces as  $n$  increases, especially for UCF-Sports and UCF-101 for which the motion overlap for a gap of  $n=10$  frames is around 60%. This implies that there is still overlap between the ground-truth boxes that are separated by  $n=10$  frames. It also means that in many cases, this overlap is below 50% due to the motion of the actor.

In practice, we want to know if we have positive training anchor cuboids. Positive cuboids are the ones that have an overlap of at least 50% with a ground-truth tubelet;

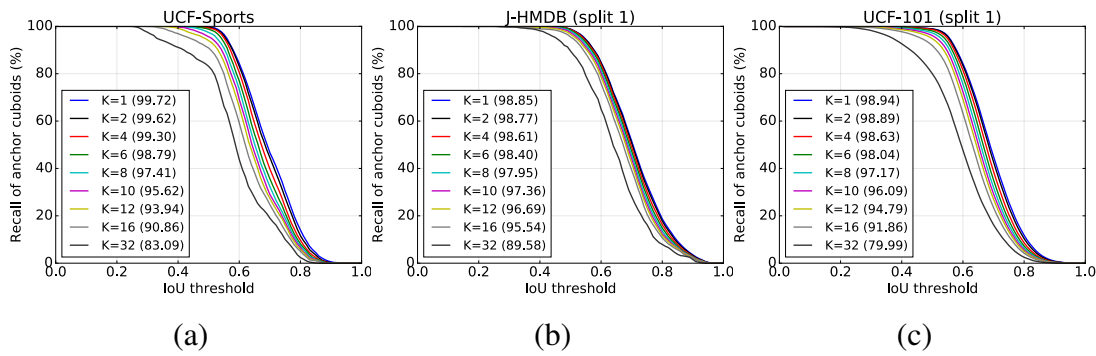


Figure 5.7: (a-c) Recall of the anchor cuboids for various IoU thresholds on the training set of three action localization datasets. The numbers in parenthesis indicate the recall at  $IoU = 0.5$ .

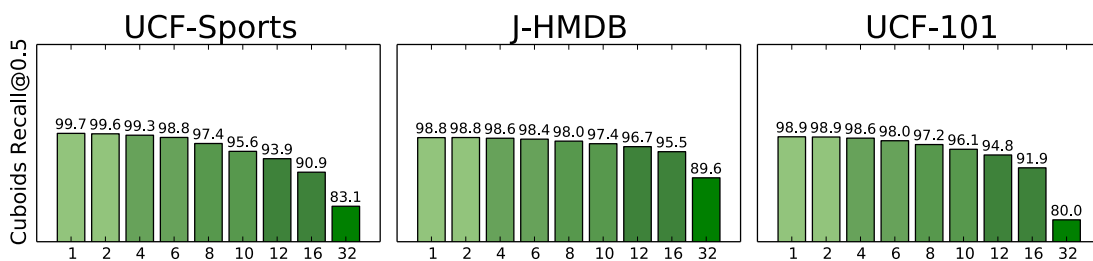


Figure 5.8: Recall of the anchor cuboids at  $IoU = 0.5$  on the training set of three action localization datasets.

the overlap being the average IoU between boxes over the  $K$  frames in the sequence. Such cuboids are required for training the classifier and the regressor. Thus, we consider all possible training sequences and compute for each class the recall of the anchor cuboids with respect to the ground-truth tubelets, i.e., the ratio of ground-truth tubelets for which at least one anchor cuboid has an overlap over 0.5.

We perform this experiment on the three action localization datasets and report the mean recall over the classes for varying IoU thresholds in Figure 5.7. For all datasets, the recall at  $IoU = 0.5$  remains  $\geq 98\%$  up to  $K = 6$  and over  $95\%$  for  $K = 10$  (Figure 5.8). This confirms that cuboid-shaped anchors can be used in case of moving actors. When increasing  $K$ , for instance to 32, the recall starts dropping significantly.

Given that sequences of up to  $K = 10$  frames results in high recall of the anchor cuboids, which is required for having positive training samples in our ACT-detector, in the following Sections 5.4.4 and 5.4.5 we examine the performance of our tubelet detector for sequences of length ranging between  $K = 1$  and  $K = 10$ .

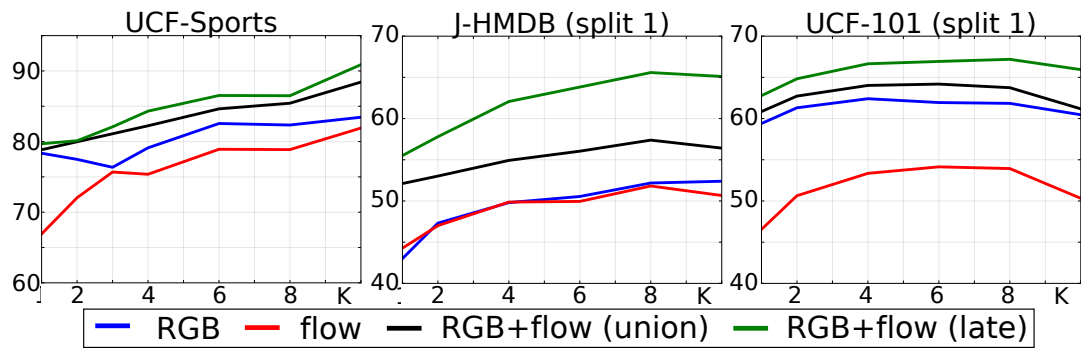


Figure 5.9: *Frame-mAP of our ACT-detector on the three datasets when varying  $K$  for RGB data (blue line), flow (red line), union and late fusion of RGB + flow data (black and green lines, resp.).*

#### 5.4.4 Tubelet modality

In this section, we examine the impact of the RGB and flow modalities and their fusion on the performance of our ACT-detector. For all datasets, we examine the frame-mAP when using (i) only RGB data, (ii) only flow data, (iii) union of RGB + flow data [Saha et al. \[2016\]](#), and (iv) late fusion of RGB + flow data for varying sequence length, ranging from 1 to 10 frames, see Figure 5.9.

For all datasets and for all  $K$ , the RGB stream (blue line) outperforms the flow stream (red line), showing that appearance information is on average a more distinctive cue than motion.

In all cases, using both modalities (green and black lines) improves the detection performance compared to using each stream separately. We observe that late fusion of the scores (green line) performs consistently better than fusion based on the union of the detections (black line), with a gain between 1% and 4% in terms of frame-mAP. This can be explained by the fact that union fusion considers a bigger set of detections without taking into account the similarity between appearance and motion detections. Instead, the late fusion re-scores every detection by taking into account both RGB and flow scores. Given that late fusion results in the best performance, we use it in the remainder of this section.

#### 5.4.5 Tubelet length

In this section, we examine the impact of  $K$ . We consider  $K = 1$  as the baseline, and we report results for our method with  $K = 2, 4, 6, 8, 10$ . We quantify the impact of  $K$  by measuring (i) the localization accuracy (MABO), (ii) the classification accuracy,

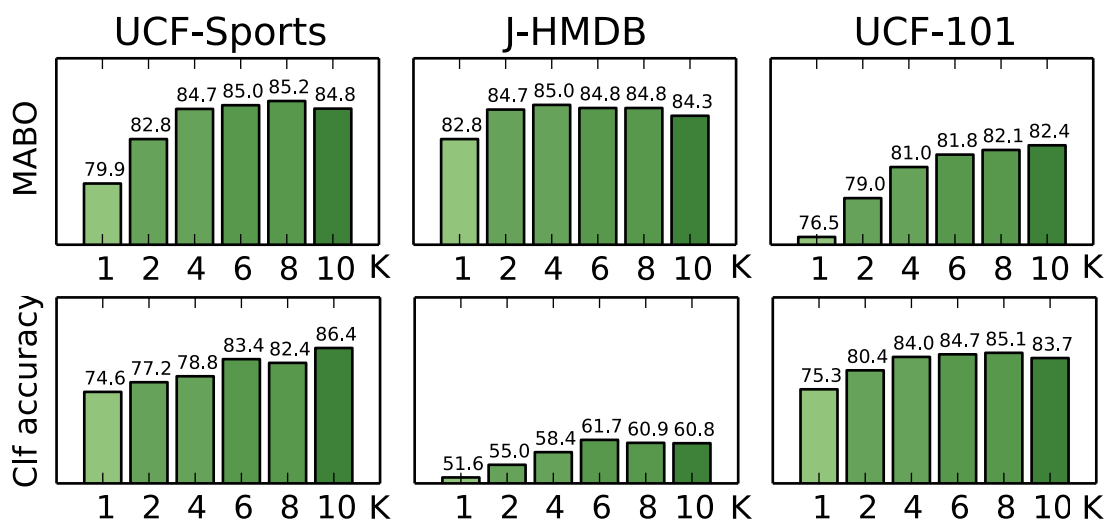


Figure 5.10: MABO (top) and classification accuracy (bottom) of our ACT-detector on the three datasets when varying  $K$ . For J-HMDB and UCF-101 we report results on the first split.

(iii) the detection performance (frame-mAP), and (iv) the motion speed of actors.

**MABO.** MABO allows us to examine the localization accuracy of the per-frame detections when varying  $K$ . Results are reported in Figure 5.10 (top). For all three datasets we observe that using sequences of frames ( $K > 1$ ) leads to a significant improvement. In particular, MABO increases up to  $K = 4$ , and then remains almost constant up to  $K = 8$  frames. For instance, MABO increases by 5% on UCF-Sports, 2% on J-HMDB and 5% on UCF-101 when using  $K = 6$  instead of  $K = 1$ . This clearly demonstrates that performing detection at the sequence level results in more accurate localization, see Figure 5.4 for examples. Overall, we observe that  $K = 6$  is one of the values for which MABO obtains excellent results for all datasets.

**Classification accuracy.** We report classification accuracy on the three action localization datasets in Figure 5.10 (bottom). Using sequences of frames ( $K > 1$ ) improves the classification accuracy of the detections for all datasets. For UCF-Sports, the accuracy keeps increasing with  $K$ , while for J-HMDB it remains almost constant after  $K = 6$ . For UCF-101, the accuracy increases when moving from  $K = 1$  to  $K = 4$  and after  $K = 8$  it starts decreasing. Overall, using up to  $K = 10$  frames improves performance over  $K = 1$ . This shows that the tubelet scoring improves the classification accuracy of the detections. Again,  $K = 6$  is one of the values which results in excellent results for all datasets.



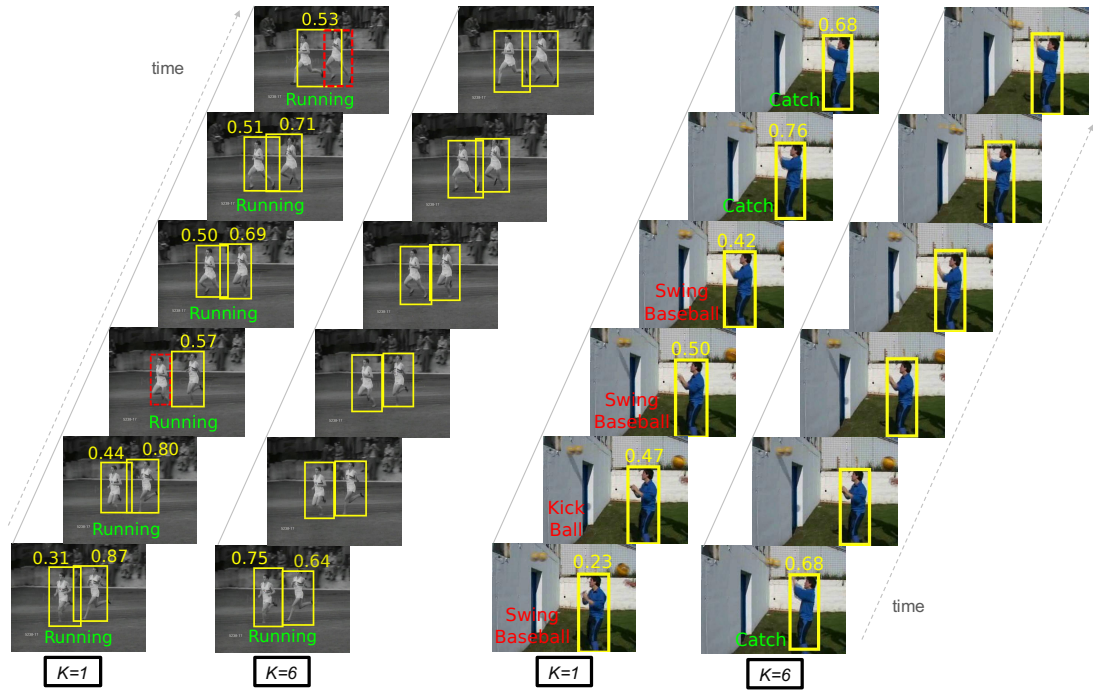


Figure 5.11: Examples when comparing per-frame ( $K=1$ ) and tubelet detections ( $K=6$ ). The yellow color represents the detections and their scores for the classes shown, the red color highlights errors either due to missed detections (first column) or wrong labeling (third column) while the green color corresponds to correct labels. Our ACT-detector outputs one class label with one score per tubelet, we thus display it once.

**Frame-mAP.** Figure 5.9 shows the frame-mAP when training the ACT-detector with varying  $K$ . On all three datasets, we observe a gain up to 10% when increasing the tubelet length up to  $K=6$  or 8 frames depending on the dataset, compared to the standard baseline of per-frame detection. This result highlights the benefit of performing detection at the sequence level. For J-HMDB and UCF-101, we also observe a performance drop for  $K > 8$ , because regressing from anchor cuboids is harder as (a) the required transformation is larger when the actor moves, and (b) there are more training parameters for less positive samples, given that the recall of anchor cuboids decreases (Section 5.4.3).

The above results show that  $K=6$  gives overall good results. We use this value in the following sections.

Figure 5.11 shows some qualitative examples comparing the performance for  $K=1$  and  $K=6$ . We observe that our tubelets lead to less missed detections and to more accurate localization compared to per-frame detection (first and second rows). Moreover, our ACT-detector reduces labeling mistakes when one frame is not enough to disam-



biguate between classes. For instance, in the last row we predict the correct label *catch*, whereas in the third row there is a big variance in the labels (*swing basketball*, *kick ball*, *catch*).

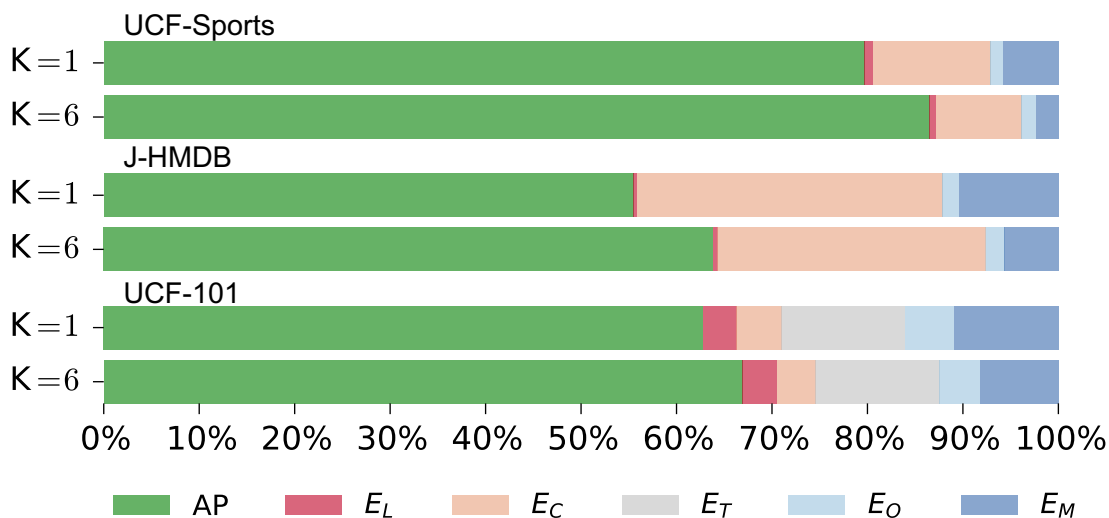


Figure 5.12: Error analysis of our ACT-detector for sequence length  $K = 1$  and  $K = 6$  on three action localization datasets. We show frame-mAP and different sources of error, see Section 5.4.6 for details. For J-HMDB and UCF-101 we report results on the first split.

### 5.4.6 Error breakdown analysis

In this section, we examine the cause of errors in frame-mAP to better understand the reasons why our tubelets improve detection performance. More precisely, we consider five mutually exclusive factors and analyze which percentage of the mAP is lost due to each of them:

1. localization error  $E_L$ : the detection is in a frame containing the correct class, but the localization is wrong, i.e.,  $IoU < 0.5$  with the ground-truth box.
2. classification error  $E_C$ : the detection has  $IoU \geq 0.5$  with the ground-truth box of another action class.
3. time error  $E_T$ : the detection is in an untrimmed video for the correct class, but the temporal extent of the action does not cover this frame.
4. other errors  $E_O$ : the detection appears in a frame without the class, and has  $IoU < 0.5$  with ground-truth boxes of any other class.
5. missed detections  $E_M$ : we do not have a detection for a ground-truth box.

| $K$     | UCF-Sports |        |      | J-HMDB (split 1) |        |      | UCF-101 (split 1) |        |      |
|---------|------------|--------|------|------------------|--------|------|-------------------|--------|------|
|         | slow       | medium | fast | slow             | medium | fast | slow              | medium | fast |
| $K = 1$ | 79.5       | 84.0   | 68.1 | 61.2             | 55.5   | 49.0 | 69.6              | 73.5   | 67.3 |
| $K = 6$ | 85.5       | 89.7   | 76.8 | 69.8             | 66.9   | 58.0 | 75.4              | 78.5   | 70.7 |

Table 5.1: Frame-mAP for slow, medium and fast moving actors.

The first four factors are categories of false positive detections, while  $E_M$  refers to the ones we did not detect at all. For the first four factors, we follow the frame-mAP computation and measure the area under the curve when plotting the percentage of each category at all recall values. The missed detections ( $E_M$ ) factor is computed by measuring the percentage of missing detections, i.e., the ratio of ground-truth boxes for which there are no correct detections.

Figure 5.12 shows the percentage that each of these factors contributes to errors in the mAP for  $K = 1$  and  $K = 6$  with late fusion of RGB and flow as input modalities. For all datasets, we observe that when going from  $K = 1$  to  $K = 6$  there is almost no change in  $E_L$  or in  $E_O$ . In particular, for UCF-Sports and J-HMDB their values are extremely small even for  $K = 1$ . We also observe a significant decrease of  $E_C$  between  $K = 1$  and  $K = 6$ , in particular on the UCF-Sports and J-HMDB datasets. This highlights that including more frames facilitates the action classification task (Figure 5.1). This drop is lower on the UCF-101 dataset. This can be explained by the fact that most errors in this dataset come from false detections outside the temporal extent of the actions ( $E_T$ ). Note that  $E_T = 0$  on UCF-Sports and J-HMDB, as these datasets are trimmed. For all datasets, a big gain comes from the missed detections  $E_M$ : for  $K = 6$  the percentage of missed detections drops significantly compared to  $K = 1$ . For instance, on J-HMDB the percentage of missed detections is reduced by a factor of 2. This clearly shows the ability of our proposed ACT-detector not only to better classify and localize ( $E_C$  and MABO) actions but also to detect actions missed by the single-frame detector (see Figure 5.11).

### 5.4.7 Handling moving actors

Our ACT-detector handles large displacements by regressing the anchor cuboids (Figures 5.3-5.7). To validate that our ACT-detector can handle moving actors, we measure frame-mAP with respect to the speed of the actor. We group actors into three categories (slow, medium, fast) with 1/3 of the data in each category. Speed is computed using

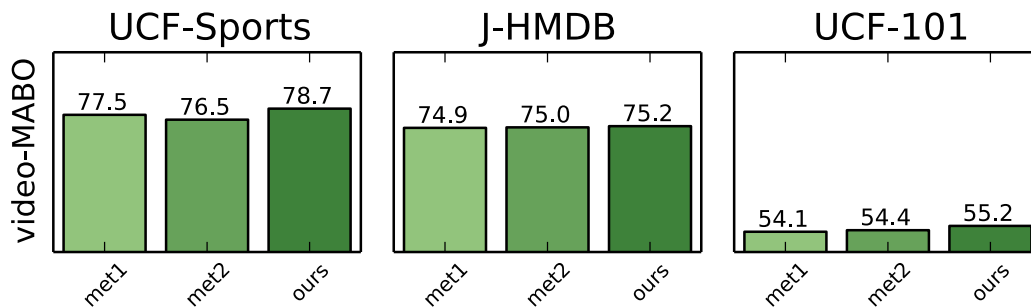


Figure 5.13: *video-MABO* of our tubelet detections with different linking methods. We employ the methods of [Peng and Schmid \[2016\]](#) (*‘method1’*), and of [Singh et al. \[2017\]](#) (*‘method2’*).

the IoU of an actor with its instances in  $\pm 10$  neighboring frames.

Table 5.1 reports the frame-mAP at IoU = 0.5 for the three categories. For all datasets there is a clear gain between  $K = 1$  and  $K = 6$  for all speeds. In particular, for actors with fast motion the gain is +8% for UCF-Sports, +9% for J-HMDB, and +3% for UCF-101. This confirms that our tubelets can successfully handle large displacements. A potential explanation is the fact that the receptive fields are significantly larger than the the spatial extent of the anchor cuboid.

#### 5.4.8 Comparison to other linking methods

To examine the quality of our tubes, we compare our linking method as described in Section 5.3.3 with other state of the art linking approaches. In particular, we deploy the linking methods of [Peng and Schmid \[2016\]](#) and [Singh et al. \[2017\]](#): these are described in paragraph [Linking and tracking methods for action localization] in Section 5.2. We use MABO and video-mAP to compare the linking methods.

| Linking method                          | UCF-Sports  |             |             |             | J-HMDB      |             |             |             | UCF-101     |             |             |             |
|---|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
|   | 0.2         | 0.5         | 0.75        | 0.5:0.95    | 0.2         | 0.5         | 0.75        | 0.5:0.95    | 0.2         | 0.5         | 0.75        | 0.5:0.95    |
| <a href="#">[Peng and Schmid, 2016]</a> | <b>92.7</b> | 91.9        | 65.1        | 53.3        | 73.9        | 72.9        | 48.7        | 42.5        | 76.1        | 50.9        | 20.6        | 23.9        |
| <a href="#">[Singh et al., 2017]</a>    | 92.6        | 87.0        | 68.7        | 52.6        | 73.8        | 72.8        | 48.8        | 42.5        | 76.6        | <b>51.5</b> | 20.5        | 24.1        |
| <b>ours</b>                             | <b>92.7</b> | <b>92.7</b> | <b>78.4</b> | <b>58.8</b> | <b>74.2</b> | <b>73.7</b> | <b>52.1</b> | <b>44.8</b> | <b>77.2</b> | 51.4        | <b>22.7</b> | <b>25.0</b> |

Table 5.2: *Comparison of different linking methods. For our tubelet detections we computed the video-mAP with various state-of-the-art linking strategies. For J-HMDB we report results averaged over all splits, and for UCF-101 we report results on the first split.*

**MABO results comparing various linking methods.** MABO allows us to examine the localization accuracy of the created tubes. Results are reported in Figure 5.13. For all three datasets, we observe that all methods have similar MABO values. Our linking method, however, for all datasets leads to slightly and consistently higher MABO values. For instance, on UCF-101, our linking method outperforms the other approaches by 0.8%. This clearly shows that our tubelet linking method results in more accurate tubes compared to other state-of-the-art methods.

**Video-mAP results comparing various linking methods.** Table 5.2 reports the video-mAP results from our tubelet detections when using different linking methods. We report video-mAP at various IoU thresholds (0.2, 0.5, and 0.75). We also report results with the protocol 0.5:0.95 of Lin et al. [2014b], which averages over multiple IoU thresholds, i.e., over 10 IoU thresholds between 0.5 and 0.95 with a step of 0.05. We observe that all methods are very close in terms of performance; yet there are some small differences. In particular, the method of Peng and Schmid [2016] results in higher video-mAP for low thresholds. For instance, on UCF-Sports at 0.5 threshold the [Peng and Schmid, 2016] achieves +3% video-mAP compared to [Singh et al., 2017] while it performs on par with ours. At higher thresholds, however, our tubelet linking method outperforms both other methods. On UCF-101, for example, our method outperforms the others by 2% IoU threshold 0.75, and by 1% at the averaged IoU threshold.

#### 5.4.9 Comparison to the state of the art

We compare our ACT-detector to the state of the art. Note that our results reported in this section are obtained by stacking 5 consecutive flow images [Peng and Schmid, 2016; Simonyan and Zisserman, 2014] as input to the motion stream, instead of just 1 for each of the  $K = 6$  input frames. This variant brings about +1% frame-mAP.

**Frame mAP.** We report frame-mAP on the three datasets in Table 5.3. We compare our performance with late fusion of RGB+5flows when  $K = 6$  to [Gkioxari and Malik, 2015; Weinzaepfel et al., 2015], that use a two-stream R-CNN, and to the method of Wang et al. [2016a], which is based on an actionness estimation. We also compare to Peng and Schmid [2016] that build upon a two-stream Faster R-CNN with multiscale training and testing. We report results of Peng and Schmid [2016] with and without their multi-region approach. The latter case can be seen as the baseline Faster R-CNN

| detector   | method                         | UCF-Sports  | J-HMDB      | UCF-101     |
|------------|--------------------------------|-------------|-------------|-------------|
| actionness | Wang et al. [2016a]            | -           | 39.9        | -           |
| R-CNN      | Gkioxari and Malik [2015]      | 68.1        | 36.2        | -           |
|            | Weinzaepfel et al. [2015]      | 71.9        | 45.8        | 35.8        |
| Faster     | Peng and Schmid [2016] w/o MR  | 82.3        | 56.9        | 64.8        |
| R-CNN      | Peng and Schmid [2016] with MR | 84.5        | 58.5        | 65.7        |
| SSD        | <b>ours</b>                    | <b>87.7</b> | <b>65.7</b> | <b>67.1</b> |

Table 5.3: Comparison of frame-mAP to the state of the art. For Peng and Schmid [2016], we report the results with and without their multi-region (+MR) approach. For J-HMDB we report results averaged over all splits, and for UCF-101 we report results on the first split.

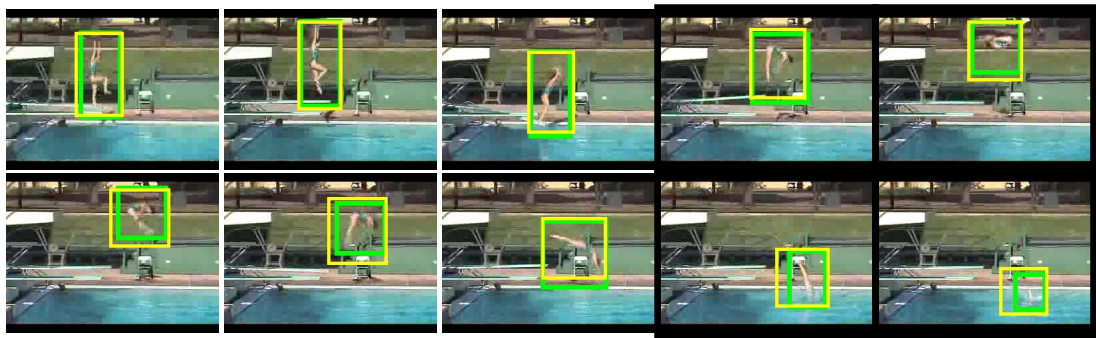
with multiscale training and testing for  $K = 1$ . Our ACT-detector (i.e., with  $K = 6$ ) allows a clear gain in frame-mAP thus, outperforming the state of the art on UCF-Sports, J-HMDB and UCF-101.

We also observe that overall, the performance of the baseline SSD ( $K = 1$ ) is somewhat lower (by around 3 to 5%), see Figure 5.9, than Faster R-CNN used by the state of the art [Peng and Schmid, 2016]. SSD, however, is much faster than Faster R-CNN, and therefore more suitable for large video datasets.

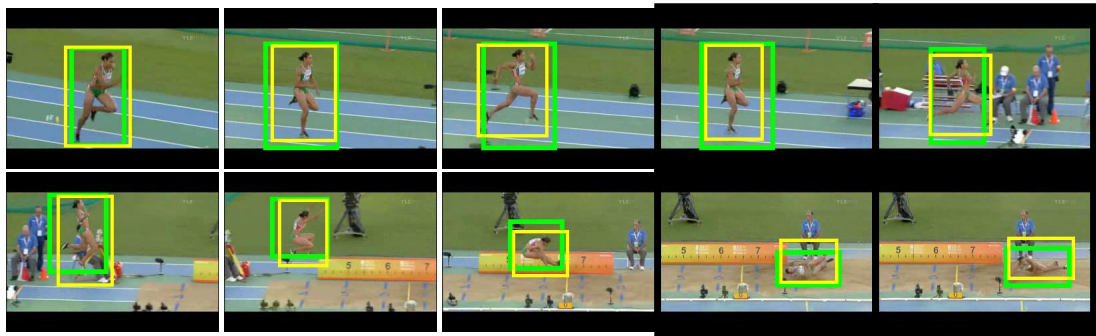
**Video mAP.** Figures 5.14-5.15 shows qualitative examples of our generated tubes: (green) ground-truth tube, (yellow) our predicted tube, (red) failure cases of our predicted tube. Figure 5.14 illustrates examples of correctly generated tubes. Some failure cases are due to the inaccurate temporal annotation (*basketball* example in Figure 5.15 (a)), whereas in other cases our method results in inaccurate spatial localization of the human—especially when considering multiple humans in a video, see *ice dancing* example in Figure 5.15 (b).

Table 5.4 reports the video-mAP results for our method and the state of the art at various IoU thresholds (0.2, 0.5, and 0.75). We also report results with the protocol 0.5:0.95 Lin et al. [2014b], which averages over multiple IoU thresholds, i.e., over 10 IoU thresholds between 0.5 and 0.95 with a step of 0.05.

At rather low IoU thresholds (0.2, 0.5), on UCF-Sports, we observe that the performance of our ACT-detector is comparable to the state of the art, whereas for higher IoU thresholds we significantly outperform Peng and Schmid [2016]. For J-HMDB, at low IoU thresholds (0.2), we perform on par with the state-of-the-art methods [Peng and



(a) diving



(b) long jump



(c) pole vault

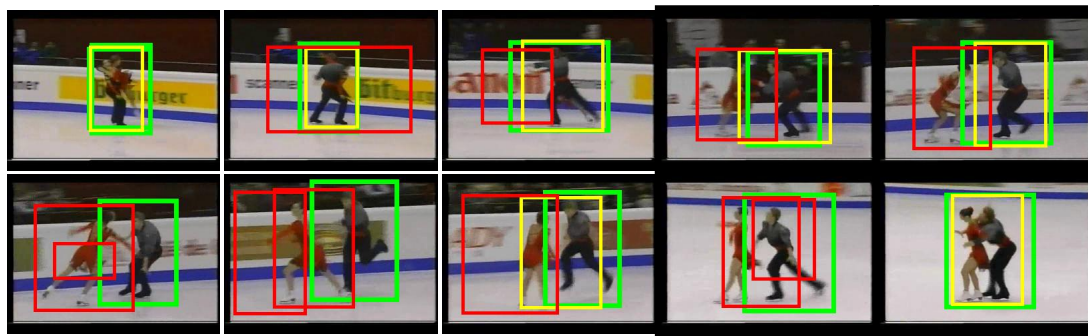
Figure 5.14: Examples of our generated tubes (yellow) with the ground-truth tubes (green) for videos of the UCF-101 [Soomro et al., 2012] dataset.

[Schmid, 2016; Saha et al., 2016] that rely on Faster R-CNN. At higher overlap thresholds we significantly outperform them. For instance on UCF-Sports and J-HMDB at  $\text{IoU} = 0.75$  we outperform ? by 31% and 4%. In particular, our performance drops slower than the state of the art as the IoU threshold increases. This highlights the high localization accuracy of our tubelets and, therefore of our tubes. On UCF-101, we significantly outperform the state of the art at all overlap thresholds, with a larger gap at high thresholds. For instance, we outperform [Singh et al., 2017] by 5% at  $\text{IoU} = 0.5$ , and by 7.5% at  $\text{IoU} = 0.75$ . As a summary, we see that our ACT-detector allows to





(a) basketball



(b) ice dancing

Figure 5.15: Failure (red) and correct cases (yellow) of our generated tubes with the ground-truth tubes (green) for videos of the UCF-101 [Soomro et al., 2012] dataset.

boost performance, especially at high thresholds.

**Runtime.** We compare our runtime using two streams (appearance and flow) to the frame-based SSD approach of Singh et al. [2017] and to frame-based Faster R-CNN approaches [Peng and Schmid, 2016; Saha et al., 2016]. We report runtime on a single GPU without flow computation. Faster R-CNN based approaches Peng and Schmid [2016]; Saha et al. [2016] run at 4fps and the SSD-based method [Singh et al., 2017] at 25-30fps. Our ACT-detector also runs at 25-30fps ( $K=6$ ). Computing tubelets has a low overhead, since the convolutional features are computed once per frame due to the parallel architecture with shared weights. The post-processing is extremely fast ( $\sim 300$ fps) for all methods.

## 5.5 Conclusions

We introduced the ACT-detector, a tubelet detector that leverages the temporal continuity of video frames. It takes as input a sequence of frames and outputs *tubelets*. This is in contrast to the previous state-of-the-art methods that operate on single frames. We

| detector     | method                    | UCF-Sports  |             |             |             | J-HMDB      |             |             |             | UCF-101     |             |             |             |
|--------------|---------------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
|              |                           | 0.2         | 0.5         | 0.75        | 0.5:0.95    | 0.2         | 0.5         | 0.75        | 0.5:0.95    | 0.2         | 0.5         | 0.75        | 0.5:0.95    |
| actionness   | Wang et al. [2016a]       | -           | -           | -           | -           | -           | 56.4        | -           | -           | -           | -           | -           | -           |
| R-CNN        | Gkioxari and Malik [2015] | -           | 75.8        | -           | -           | -           | 53.3        | -           | -           | -           | -           | -           | -           |
|              | Weinzaepfel et al. [2015] | -           | 90.5        | -           | -           | 63.1        | 60.7        | -           | -           | 51.7        | -           | -           | -           |
| Faster R-CNN | Peng and Schmid [2016]:   |             |             |             |             |             |             |             |             |             |             |             |             |
|              | w/o MR                    | <b>94.8</b> | <b>94.8</b> | 47.3        | 51.0        | 71.1        | 70.6        | 48.2        | 42.2        | 71.8        | 35.9        | 1.6         | 8.8         |
|              | with MR                   | <b>94.8</b> | 94.7        | -           | -           | <b>74.3</b> | 73.1        | -           | -           | 72.9        | -           | -           | -           |
|              | Saha et al. [2016]        | -           | -           | -           | -           | 72.6        | 71.5        | 43.3        | 40.0        | 66.7        | 35.9        | 7.9         | 14.4        |
| SSD          | Singh et al. [2017]       | -           | -           | -           | -           | 73.8        | 72.0        | 44.5        | 41.6        | 73.5        | 46.3        | 15.0        | 20.4        |
|              | <b>ours</b>               | 92.7        | 92.7        | <b>78.4</b> | <b>58.8</b> | 74.2        | <b>73.7</b> | <b>52.1</b> | <b>44.8</b> | <b>77.2</b> | <b>51.4</b> | <b>22.7</b> | <b>25.0</b> |

Table 5.4: Comparison of video-mAP to the state of the art at various detection thresholds. The columns 0.5:0.95 correspond to the average video-mAP for thresholds in this range. For Peng and Schmid [2016], we report the results with and without their multi-region (+MR) approach. For J-HMDB we report results averaged over all splits, and for UCF-101 we report results on the first split.

build our method upon SSD relying on anchor cuboids. We extract convolutional features for all frames and stack these feature maps over time. Then, we score and regress the anchor cuboids, exploiting the temporal information over sequences of frames. Our extensive experimental analysis highlights the benefits of our ACT-detector for both classification and localization. Our ACT-detector achieves state-of-the-art results for both frame-mAP and video-mAP on modern action localization datasets, in particular for high overlap thresholds. In Appendix E we report per-class results for the experiments of this chapter.





# Chapter 6

## Conclusions

### Contents

---

|   |            |
|---|------------|
| <b>6.1 Summary of contributions . . . . .</b>         | <b>147</b> |
| <b>6.2 Future research and perspectives . . . . .</b> | <b>150</b> |

---

**I**n this thesis we have focused on three tasks of video understanding: object detection, joint object and action detection, and spatio-temporal action localization. Our goal was to create techniques for analyzing videos, either by relying on detectors adapted from images to the videos or by exploiting the spatio-temporal continuity of actions present in videos.

**Outline.** This chapter is organized as follows. Section 6.1 summarizes the contributions of the thesis and Section 6.2 gives directions for future research inspired from the work done in the thesis.

### 6.1 Summary of contributions

In this section, we summarize our contributions.

**The first contribution** described in Chapter 3 is examining how to leverage videos for object detection. To do so, we explored the differences between videos and images for object detection. We casted these differences as five domain shift factors: spatial location accuracy, appearance diversity, image quality, aspect distribution, and camera framing. Following a systematic protocol, we thoroughly analyzed these factors first by measuring each one of them in two image and two video detection datasets, and then

by examining their impact on the performance of object detectors. We showed that by progressively canceling out these factors we gradually close the performance gap between training on the test domain and training on the other domain. Our experiments led to several useful conclusions:

1. training from videos with ground-truth bounding box annotations still produces a worse detector than when training from still images. Hence, future research on video segmentation cannot solve the problem *on its own*;
2. image quality and especially video blur has a strong impact on the performance gap;
3. the appearance diversity and aspect distribution of a training set is much more important than the number of training samples it contains: for good performance one should collect a broad range of videos showing all aspects expected to appear in the test set.

However, this contribution presents some limitations. Our exploration includes training data coming only from one domain—either images or videos. Instead, one could also examine the detection performance when combining image and video training data. In Appendix C we present an initial attempt of using both image and video training sets. However, this remains an unexplored field, as there are various ways of combining training data from both domains using domain adaptation techniques.

Moreover, our exploration is limited to object detection datasets and is not generalizable to action datasets, as we cannot examine the actions on still images. Some of our findings though can be useful for action detection. For instance, the fact that not perfect spatial annotations produce only slightly worse detectors than the perfect ones do can be helpful for building action datasets, as we showed perfect spatial annotations for all frames are not always necessary.

**The second contribution** described in Chapter 4 is a method for joint object and action detection in uncontrolled video scenes, e.g., *car rolling*. This was in contrast to state-of-the-art approaches that focus either on objects or on actions. In particular, we exploited the motion of videos to understand the action of moving objects or actors, given that the temporal continuity of videos entails unique characteristics related to motion cues. We found that detecting objects and actions jointly (instead of separately) boosts the detection performance of each task as the two tasks help each other. We

proposed an end-to-end network with two streams for appearance and motion, see Section 4.3. The appearance stream uses RGB frames, while the motion stream uses optical flow between consecutive video frames, thus exploiting the temporal continuity of videos. We casted the joint training as a multitask objective, that

1. outperforms training alone either with objects or with actions, as the network is better able to generalize, less prone to overfit, and benefits from sharing statistical strength between classes,
2. performs as well as other variants, such as Cartesian or hierarchical combination of objects and actions, while requiring fewer parameters,
3. allows zero-shot learning of actions performed by an object, for which no action labels are present at training time, and
4. makes our architecture a generalized building block for different tasks, such as semantic segmentation or visual relationships between objects.

This method has also a few limitations. It is limited in detecting object-action pairs only spatially, as we do not include any temporal detection. Nevertheless, real-world systems could benefit by adding temporal detection, as it is also important to detect *when* a car is crashing or a baby is crying.

**The third contribution** described in Chapter 5 consists in introducing the ACT-detector, a novel tubelet detector that takes as input a sequence of frames and outputs a sequence of regressed boxes with their associated scores, i.e., *tubelets*, see Section 5.3. In contrast to Chapters 3-4 where we had detected moving objects and actions using *only* single (for appearance) and consecutive (for motion) frames, our ACT-detector considers longer temporal analysis of videos by using sequences of frames. We built our ACT-detector relying on both appearance and motion cues. By scoring and regressing spatio-temporal cuboids, our tubelet detector better exploits the temporal information of frames. We demonstrated that

1. performing action localization for a long sequence of frames, instead of a pair of subsequent frames, improves the detection performance,
2. our proposed anchor cuboids handle large displacements,
3. our ACT-detector better localizes and classifies the actions in videos, and

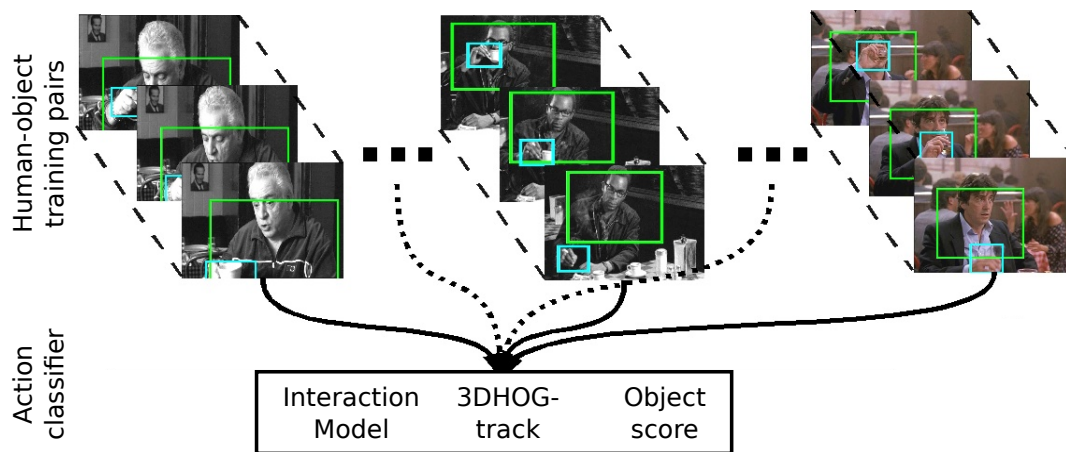


Figure 6.1: Example of pipeline that models interactions between humans and objects. Image from [Prest et al., 2013].

4. our detection pipeline outperforms all other state-of-the-art methods on action localization, especially for high overlap thresholds.

Our method is limited in detecting tubelets only of fixed-length. Ideally, the length of the tubelets should vary. For instance, a *human running* and a *human playing golf* do not have the same motion speed, and therefore the length of their tubelets should also be different. Moreover, our linking method depends only on hand-crafted features, such as the bounding box coordinates or the overlap of bounding boxes. Another approach would be to employ a method that *learns* how to link tubelets over time by using the aforementioned hand-crafted features or by also considering the appearance features between consecutive tubelets.

## 6.2 Future research and perspectives

In this section, we propose directions for future research based on the experiments presented in this thesis as well as recent advances in the field of computer vision and machine learning.

**Exploiting various cues.** Initialization is crucial for CNNs, especially when the amount of training data is limited. Some works have used models pre-trained on videos either with [Gkioxari and Malik, 2015; Simonyan and Zisserman, 2014; Weinzaepfel et al., 2015] or without [Wang and Gupta, 2015] bounding box annotations. Nevertheless, the diversity of current video datasets is limited, and hence pre-training on videos

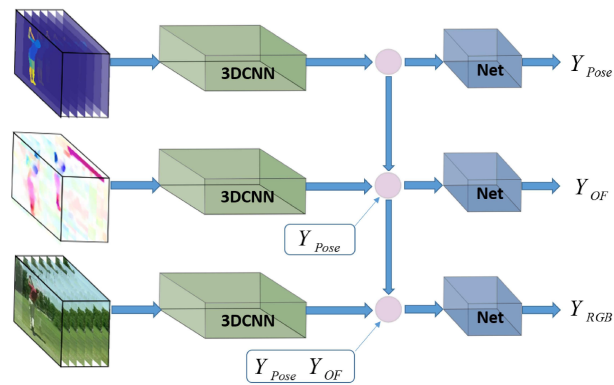


Figure 6.2: *Three-stream network for action classification: RGB, optical flow and pose.* Image from [Zolfaghari et al., 2017].

produces worse classifiers than pre-training on images. For this reason, most modern CNN methods use models pre-trained on images for the classification task [Krizhevsky et al., 2012]. An interesting line of work would be to explore the impact of different initializations on the detection performance. For instance, action localization could benefit from initialization with human tubes [Weinzaepfel et al., 2017] or from large video datasets Carreira and Zisserman [2017].

High-level information, such as human poses [Zolfaghari et al., 2017], segmentation masks [Huang et al., 2016b], or deformable parts [Modolo and Ferrari, 2016], provides a richer representation of the actors, and hence can help action recognition. For instance, the pose of humans can be meaningful for action recognition (see Figure 6.2), and in particular for identifying the beginning or the end of a video action. An interesting extension of this would be to integrate motion of the pose over time into a 3D convolutional architecture, as for videos it has been proven useful to shift towards the 3D domain, where the third dimension is the time Carreira and Zisserman [2017].

Typically the actors or objects in videos interact with each other [Prest et al., 2013; Xu et al., 2015]. An interesting perspective is to model the interactions between moving objects or actors over time, see Figure 6.1. There are some works that model such interactions [Prest et al., 2012b, 2013], but only on limited settings, e.g., small datasets with only a few object classes. Using recent advances in video detection and tracking can help generalizing to realistic video datasets.

Another interesting category is to enhance video information with audio features. Audio can facilitate the classification of video-level actions, such as *talking* or *listening to music*, and also can assist the temporal localization, e.g., *kicking ball* or *opening door*.

**Tube detector.** Motion is an intrinsic characteristic of videos, and therefore, object or action detection in videos can benefit from this information. Recent works on video detection exploit the video motion by including optical flow [Peng and Schmid, 2016; Singh et al., 2017; Saha et al., 2016] or even human pose [Zolfaghari et al., 2017] in the training models. Albeit their results, these methods do not truly exploit the temporal continuity of videos: distinguishing actions even with the pose from a single frame can be ambiguous, e.g., *person getting into or out of a vehicle*. In Chapter 5, we built an action detector that uses sequences of frames to output fixed-length tubelets, which are then linked over time to create tubes. On top of this, there are many future directions. For instance, the length of tubelets is limited by the human motion, in particular when tubelets are obtained by regressing cuboids. Instead of fixed-length units (either frames or sequences of frames), one interesting idea is to detect tubelets of varying length depending on the human motion at this particular instance of the video. Another line of work is including tubelet linking in an end-to-end architecture, which would first detect spatio-temporal tubelets, and then link them over time. To this end, one could leverage LSTMs, which after being trained jointly with the detection network can produce tubes of variable length. A further and ideal extension includes directly localizing tubes spanning across the whole action.

As actions do not necessarily start or end at the beginning or end of videos, one should also consider temporal detections. Identifying the temporal window where the action actually occurs is challenging [Caba Heilbron et al., 2016; Escorcia et al., 2016], especially for long actions. One solution is to measure the relative duration of an action or the *actionness* of tubelets using the training sets and discard tubelets with low actionness score. Inspired by activity recognition [Heilbron et al., 2015] (Figure 6.3), another solution is to decompose temporally actions into sub-actions. Then, after temporally localizing sub-actions, one can merge them to create the final action.

**Weakly-supervised learning.** Training object detectors requires manually annotating with bounding boxes a big pool of images. The annotation task is very time consuming and expensive, as it requires about half a minute per bounding box [Su et al., 2012]. A way to reduce this annotation time is the weakly-supervised learning, where the detectors are trained from a set of images labeled only as containing a certain object class, without being given the location of the objects. The goal is to localize the objects in these training images while learning an object detector for detecting instances in new test images. While this setting is substantially cheaper, the resulting detectors





initialized using key-frame extraction methods, such as video summarization [Zhang et al., 2016].

Annotating instances in images is expensive; annotating instances in videos is orders of magnitude above images, as video datasets contain millions of frames that require both spatial and temporal annotations. The annotations can be bounding boxes, as in UCF-101 [Soomro et al., 2012] with half a million of annotated frames or even pixel-level annotations, as in the recently-proposed DAVIS dataset [Perazzi et al., 2016], which contains pixel-level and attribute-level annotations for a few thousands frames. To reduce this enormous annotation cost, some recent approaches tackle a slight modified weakly-supervised setting in videos, as they rely on weakly-labeled videos with some additional annotations. For instance, Mathe and Sminchisescu [2012] use eye-tracking video data for action classification, Mettes et al. [2016] use only sparse-click annotations to learn action proposals for action localization, [Weinzaepfel et al., 2017] use one box annotation together with time annotation, and Manen et al. [2017] use path annotations for object tracking. Their results are promising as they reduce the annotation time and perform reasonably well.

Nevertheless, video data is becoming abundant. Thus, fully, sparsely, or even weakly-annotated datasets are not always realistic, as annotating all video datasets is very expensive. Therefore, learning unsupervised models for localizing spatially and temporally video actions is becoming important. To do so, one could first localize spatially humans in videos, for example using human tubes [Weinzaepfel et al., 2017], and then, for identifying the action class of the video, adapt models trained from other action datasets to the existing dataset using domain adaptation techniques. For temporal detection, one could either apply a sliding window approach, which finds the temporal window most likely to contain the action [Peng and Schmid, 2016; Weinzaepfel et al., 2015], or even adapt temporal networks trained on different datasets [Caba Heilbron et al., 2016; Escorcia et al., 2016].

# Publications

This thesis has led to the publications summarized below.

## International conferences

- Vicky Kalogeiton, Philippe Weinzaepfel, Vittorio Ferrari and Cordelia Schmid, Action Tubelet Detector for Spatio-Temporal Action Localization, Published to *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- Vicky Kalogeiton, Philippe Weinzaepfel, Vittorio Ferrari and Cordelia Schmid, Joint learning of object and action detectors, Published to *IEEE International Conference on Computer Vision (ICCV)*, 2017.

## International journals

- Vicky Kalogeiton, Vittorio Ferrari and Cordelia Schmid, Analysing domain shift factors between videos and images for object detection, Published to *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2016.



# Appendix A

## YouTube-Objects dataset v2.0

The YouTube-Objects dataset (YTO) was first introduced by [Prest et al. \[2012a\]](#). It contains videos collected from YouTube for 10 classes of moving objects: *aeroplane, bird, boat, car, cat, cow, dog, horse, motorbike, and train*. It consists of 155 videos; 720,152 frames; and 1,258 bounding box annotations.

In this section we present the second version of YTO. First, we split the videos into disjoint training and test sets; frames from the same video belong only to one set. We also perform shot boundary detection. Table A.1 reports these statistics per object class.

Table A.2 reports the sampling and annotation statistics for YTO v2.0. For both sets we uniformly sample a constant number of frames in each shot. Out of the pool of sampled frames, we only annotate with bounding boxes the ones that actually contain the object of interest. In particular, for the training set we annotate one object instance per frame, while for the test set we annotate all instances. In total we annotate 6,087 frames, out of 6,908 sampled frames, resulting in 6,973 annotated instances. The dataset is available online at <http://calvin.inf.ed.ac.uk/datasets/youtube-objects-dataset/>; we presented it in [[Kalogeiton et al., 2016](#)]. For examples of the annotated object instances see Figure 3.2.

| classname | # videos | # train videos | # test videos | # shots | # frames |
|-----------|----------|----------------|---------------|---------|----------|
| aeroplane | 13       | 7              | 6             | 482     | 79483    |
| bird      | 16       | 10             | 6             | 175     | 34517    |
| boat      | 17       | 12             | 5             | 191     | 119448   |
| car       | 9        | 6              | 3             | 212     | 27607    |
| cat       | 21       | 15             | 6             | 245     | 59822    |
| cow       | 11       | 9              | 2             | 70      | 41158    |
| dog       | 24       | 16             | 8             | 217     | 86306    |
| horse     | 15       | 11             | 4             | 151     | 70392    |
| motorbike | 14       | 9              | 5             | 444     | 68421    |
| train     | 15       | 11             | 4             | 324     | 132998   |
| Total     | 155      | 106            | 49            | 2511    | 720152   |

Table A.1: Statistics of videos and frames in the YTO v2.0.

| classname | # sampled frames<br>per shot | # sampled<br>frames | # frames with<br>annotations | # training<br>images/instances | # test |           |
|-----------|------------------------------|---------------------|------------------------------|--------------------------------|--------|-----------|
|           |                              |                     |                              |                                | images | instances |
| aeroplane | 2                            | 656                 | 585                          | 415                            | 170    | 180       |
| bird      | 3                            | 596                 | 514                          | 359                            | 155    | 162       |
| boat      | 2                            | 573                 | 501                          | 357                            | 144    | 234       |
| car       | 3                            | 1337                | 1278                         | 915                            | 363    | 606       |
| cat       | 1 - 2                        | 528                 | 467                          | 326                            | 141    | 165       |
| cow       | 9                            | 495                 | 461                          | 321                            | 140    | 315       |
| dog       | 4                            | 767                 | 618                          | 454                            | 164    | 173       |
| horse     | 5 - 6                        | 670                 | 608                          | 427                            | 181    | 463       |
| motorbike | 1 - 2                        | 656                 | 525                          | 360                            | 165    | 213       |
| train     | 1 - 2                        | 630                 | 530                          | 372                            | 158    | 158       |
| Total     | -                            | 6908                | 6087                         | 4306                           | 1781   | 2669      |

Table A.2: Statistics of annotations in the YTO v2.0.

# Appendix B

## Additional experimental results of domain shift factors

**I**n this appendix we present the per-class results for all experiments of Chapter 3. We report the mAP results before and after equalizing each domain shift factor: spatial location accuracy (SLA), appearance diversity (AD), image quality (IQ) — Gaussian blur (IQ-G), motion blur (IQ-M), and aspect distribution (AS). We also report the performance gap at each equalization step.

First, we report results on the first dataset pair: YTO-VOC. Table B.1 reports the number of object samples per class before each equalization step. Tables B.2-B.3 report the per-class mAP results: Tables B.2-B.3 show the mAP of the DPM and R-CCN detectors when testing on VOC (Figure 3.4), while Tables B.4-B.5 report the mAP results for the same detectors when testing on YTO (Figure 3.5).

Then, we report results on the second dataset pair: IMG and VID from ILSVRC 2015. Table B.6 reports the number of object samples per class before each equalization step. Tables B.7-B.8 report the per-frame mAP results of the R-CCN detector when testing on IMG (Figure 3.13 (a)) and when testing on VID (Figure 3.13 (b)).

| # samples for the VOC-YTO |      |      |      |     |
|---------------------------|------|------|------|-----|
| classname                 | SLA  | AD   | IQ   | AS  |
| aeroplane                 | 306  | 244  | 244  | 93  |
| bird                      | 359  | 123  | 123  | 15  |
| boat                      | 290  | 138  | 138  | 15  |
| car                       | 915  | 310  | 310  | 78  |
| cat                       | 326  | 249  | 249  | 75  |
| cow                       | 259  | 90   | 90   | 15  |
| dog                       | 454  | 295  | 295  | 79  |
| horse                     | 362  | 286  | 286  | 70  |
| motorbike                 | 339  | 243  | 243  | 43  |
| train                     | 297  | 223  | 223  | 68  |
| Total                     | 3907 | 2201 | 2201 | 551 |

Table B.1: Number of training object samples for the VOC-YTO dataset pair before cancelling out each factor.

| mAP of the DPM detector when testing on VOC |      |      |      |      |      |      |      |      |      |      |
|---|------|------|------|------|------|------|------|------|------|------|
| classname                                   | SLA  |      |      |      | AD   |      | IQ-G |      | IQ-M |      |
|   | PRE  | FVS  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  |
| aeroplane                                   | 14.7 | 20.6 | 19.1 | 30.9 | 16.0 | 29.1 | 16.0 | 28.6 | 16.0 | 25.7 |
| bird  | 9.39 | 9.77 | 9.66 | 0.89 | 9.68 | 0.83 | 9.68 | 0.57 | 9.68 | 2.42 |
| boat  | 9.19 | 7.57 | 4.93 | 11.9 | 6.79 | 10.7 | 6.79 | 9.72 | 6.79 | 9.49 |
| car   | 33.2 | 46.6 | 45.5 | 52.7 | 43.8 | 50.0 | 43.8 | 50.0 | 43.8 | 46.5 |
| cat   | 11.2 | 1.78 | 2.12 | 18.0 | 3.83 | 14.3 | 3.83 | 9.48 | 3.83 | 5.85 |
| cow   | 9.19 | 7.98 | 13.6 | 24.9 | 11.2 | 14.1 | 11.2 | 14.1 | 11.2 | 9.94 |
| dog   | 9.27 | 10.7 | 10.8 | 6.9  | 10.9 | 6.57 | 10.9 | 10.9 | 10.9 | 5.79 |
| horse                                       | 10.3 | 33.1 | 31.7 | 57.8 | 37.3 | 54.6 | 37.3 | 52.2 | 37.3 | 52.6 |
| motorbike                                   | 25.0 | 21.1 | 22.8 | 45.0 | 22.5 | 42.0 | 22.5 | 39.7 | 22.5 | 36.8 |
| train                                       | 6.86 | 20.7 | 16.3 | 45.5 | 18.5 | 39.9 | 18.5 | 35.6 | 18.5 | 36.8 |
| avg   | 13.8 | 18.0 | 17.6 | 29.4 | 18.0 | 26.2 | 18.0 | 25.1 | 18.0 | 23.2 |
| performance gap                             | 15.6 | 11.4 | 11.7 | -    | 8.16 | -    | 7.03 | -    | 5.13 | -    |

Table B.2: Per-class mAP results of the DPM detector when testing on VOC.



| mAP of the R-CNN detector when testing on VOC |      |      |      |      |      |      |      |      |      |      |      |      |
|---|------|------|------|------|------|------|------|------|------|------|------|------|
| classname                                     | SLA  |      |      |      | AD   |      | IQ-G |      | IQ-M |      | AS   |      |
|   | PRE  | FVS  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  |
| aeroplane                                     | 33.6 | 36.2 | 37.7 | 57.7 | 40.1 | 58.3 | 40.1 | 44.3 | 40.1 | 45.5 | 44.9 | 46.5 |
| bird  | 14.0 | 22.3 | 21.1 | 41.7 | 21.6 | 38.3 | 21.6 | 34.9 | 21.6 | 26.4 | 16.1 | 20   |
| boat  | 14.7 | 22.1 | 19.5 | 31.9 | 20.7 | 30.5 | 20.7 | 25.0 | 20.7 | 20.0 | 9.73 | 17.9 |
| car   | 28.1 | 41.5 | 46.8 | 59.6 | 47.1 | 55.2 | 47.1 | 50.3 | 47.1 | 50.2 | 46.3 | 48.6 |
| cat   | 20.3 | 29.6 | 35.1 | 49.0 | 36.0 | 47.0 | 36.0 | 40.2 | 36.0 | 36.4 | 38.4 | 33.1 |
| cow   | 11.0 | 16.9 | 15.6 | 39.6 | 15.8 | 34.7 | 15.8 | 24.5 | 15.8 | 28.3 | 15.9 | 17.1 |
| dog   | 16.6 | 27.0 | 19.8 | 44.9 | 21.5 | 41.7 | 21.5 | 34.3 | 21.5 | 30.4 | 20.9 | 26.2 |
| horse   | 13.0 | 16.4 | 23.4 | 46.5 | 25.6 | 46.6 | 25.6 | 43.3 | 25.6 | 40.0 | 29.7 | 32.2 |
| motorbike                                     | 24.0 | 27.6 | 35.5 | 53.1 | 36.6 | 52.3 | 36.6 | 43.8 | 36.6 | 45.3 | 39.9 | 37.0 |
| train   | 25.7 | 35.1 | 35.3 | 50.4 | 38.4 | 49.1 | 38.4 | 43.8 | 38.4 | 44.4 | 39.0 | 41.9 |
| avg   | 20.1 | 27.5 | 29.0 | 47.5 | 30.3 | 45.4 | 30.3 | 38.4 | 30.3 | 36.7 | 30.1 | 32.0 |
| performance gap                               | 27.3 | 19.9 | 18.4 | -    | 15.0 | -    | 8.08 | -    | 6.35 | -    | 1.96 | -    |

Table B.3: *Per-class mAP results of the R-CNN detector when testing on VOC.*

| mAP of the DPM detector when testing on YTO |      |      |      |      |      |      |      |      |      |      |
|---|------|------|------|------|------|------|------|------|------|------|
| classname                                   | SLA  |      |      |      | AD   |      | IQ-G |      | IQ-M |      |
|   | PRE  | FVS  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  |
| aeroplane                                   | 19.6 | 29.7 | 25.4 | 30.7 | 28.4 | 32.9 | 28.4 | 32.6 | 28.4 | 34.0 |
| bird  | 42.8 | 44.4 | 44.0 | 10.4 | 48.1 | 2.23 | 48.1 | 2.90 | 48.1 | 21.2 |
| boat  | 9.53 | 18.0 | 33.4 | 0.97 | 25.5 | 1.79 | 25.5 | 4.80 | 25.5 | 1.45 |
| car   | 35.7 | 52.3 | 49.1 | 48.6 | 48.9 | 46.5 | 48.9 | 47.9 | 48.9 | 46.6 |
| cat   | 14.4 | 5.57 | 2.72 | 18.3 | 1.69 | 12.6 | 1.69 | 16.0 | 1.69 | 10.2 |
| cow   | 13.5 | 15.0 | 20.1 | 33.6 | 19.2 | 25.0 | 19.2 | 24.9 | 19.2 | 11.5 |
| dog   | 10.3 | 13.2 | 12.4 | 13.6 | 15.8 | 2.11 | 15.8 | 1.76 | 15.8 | 0.88 |
| horse                                       | 11.3 | 35.3 | 35.1 | 26.7 | 35.1 | 25.6 | 35.1 | 26.4 | 35.1 | 22.7 |
| motorbike                                   | 32.3 | 29.4 | 29.5 | 35.8 | 31.6 | 33.6 | 31.6 | 31.8 | 31.6 | 26.2 |
| train                                       | 24.9 | 51.8 | 50.0 | 23.9 | 39.5 | 22.8 | 39.5 | 19.4 | 39.5 | 28.1 |
| avg   | 21.4 | 29.5 | 30.2 | 24.3 | 29.4 | 20.5 | 29.4 | 20.8 | 29.4 | 20.3 |
| performance gap                             | 8.8  | 0.7  |      | 5.9  | -    | 8.8  | -    | 8.5  | -    | 9.0  |

Table B.4: *Per-class mAP results of the DPM detector when testing on YTO.*

| mAP of the R-CNN detector when testing on YTO |      |      |      |      |      |      |      |      |      |      |      |      |
|---|------|------|------|------|------|------|------|------|------|------|------|------|
| classname                                     | SLA  |      |      |      | AD   |      | IQ-G |      | IQ-M |      | AS   |      |
|   | PRE  | FVS  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  | YTO  | VOC  |
| aeroplane                                     | 50.1 | 50.9 | 57.5 | 53.6 | 59.6 | 53.4 | 59.6 | 59.6 | 59.6 | 59.5 | 61.4 | 57.1 |
| bird  | 51.5 | 53.1 | 78.0 | 74.7 | 76.6 | 72.0 | 76.6 | 70.4 | 76.6 | 60.4 | 65.4 | 62.5 |
| boat  | 25.7 | 42.4 | 51.6 | 43.5 | 51.2 | 38.7 | 51.2 | 32.8 | 51.2 | 28.8 | 20.3 | 23.0 |
| car   | 38.7 | 51.6 | 55.5 | 55.4 | 55.1 | 51.3 | 55.1 | 53.9 | 55.1 | 53.7 | 55.8 | 53.0 |
| cat   | 19.4 | 29.6 | 32.7 | 35.6 | 35.6 | 32.5 | 35.6 | 32.1 | 35.6 | 33.2 | 34.9 | 32.6 |
| cow   | 11.9 | 23.0 | 28.6 | 43.7 | 25.0 | 39.7 | 25.0 | 38.8 | 25.0 | 34.3 | 23.5 | 34.1 |
| dog   | 7.73 | 24.3 | 34.6 | 24.2 | 36.4 | 19.9 | 36.4 | 21.7 | 36.4 | 16.2 | 25.2 | 22.5 |
| horse   | 7.27 | 25.0 | 26.8 | 28.0 | 27.9 | 25.6 | 27.9 | 26.8 | 27.9 | 27.3 | 27.0 | 25.1 |
| motorbike                                     | 41.1 | 48.9 | 55.1 | 54.2 | 56.1 | 53.6 | 56.1 | 52.6 | 56.1 | 45.3 | 54.1 | 42.5 |
| train   | 6.78 | 23.7 | 40.4 | 18.4 | 39.4 | 18.6 | 39.4 | 19.9 | 39.4 | 19.5 | 20.7 | 20.8 |
| avg   | 26.0 | 37.2 | 46.1 | 43.1 | 46.3 | 40.5 | 46.3 | 40.9 | 46.3 | 37.8 | 38.8 | 37.3 |
| performance gap                               | 20.1 | 8.9  | -    | 2.9  | -    | 5.7  | -    | 5.4  | -    | 8.4  | -    | 1.4  |

Table B.5: Per-class mAP results of the R-CNN detector when testing on YTO.

| # samples for the ILSVRC VID-IMG |       |      |      |      |
|----------------------------------|-------|------|------|------|
| classname                        | SLA   | AD   | IQ   | AS   |
| aeroplane                        | 225   | 173  | 173  | 63   |
| bird                             | 2824  | 1466 | 1466 | 580  |
| boat                             | 665   | 247  | 247  | 94   |
| car                              | 3513  | 1787 | 1787 | 688  |
| cat                              | 661   | 389  | 389  | 229  |
| cow                              | 299   | 194  | 194  | 24   |
| dog                              | 2860  | 2340 | 2340 | 1403 |
| horse                            | 890   | 457  | 457  | 71   |
| motorbike                        | 1185  | 665  | 665  | 147  |
| train                            | 213   | 184  | 184  | 100  |
| Total                            | 13335 | 7902 | 7902 | 3399 |

Table B.6: Number of training object samples for the ILSVRC VID-IMG dataset pair before cancelling out each factor.

| mAP of the R-CNN detector when testing on IMG |      |      |      |      |      |      |      |      |      |      |
|---|------|------|------|------|------|------|------|------|------|------|
| classname                                     | SLA  |      | AD   |      | IQ-G |      | IQ-M |      | AS   |      |
|   | VID  | IMG  | VID  | IMG  | VID  | IMG  | VID  | IMG  | VID  | IMG  |
| aeroplane                                     | 33.1 | 45.2 | 31.9 | 44.2 | 31.9 | 31.3 | 31.9 | 32.2 | 29.7 | 34.3 |
| bird  | 42.0 | 53.2 | 42.0 | 52.0 | 42.0 | 47.5 | 42.0 | 47.2 | 42.8 | 45.5 |
| boat  | 24.8 | 31.5 | 23.6 | 29.0 | 23.6 | 25.8 | 23.6 | 24.2 | 21.5 | 21.6 |
| car   | 37.4 | 39.0 | 37.3 | 36.6 | 37.3 | 34.0 | 37.3 | 34.9 | 35.5 | 33.7 |
| cat   | 31.1 | 36.9 | 32.2 | 37.0 | 32.2 | 30.2 | 32.2 | 26.8 | 30.3 | 30.3 |
| cow   | 10.6 | 19.5 | 10.3 | 19.1 | 10.3 | 21.3 | 10.3 | 5.3  | 7.0  | 20.8 |
| dog   | 44.3 | 64.3 | 45.3 | 64.3 | 45.3 | 52.4 | 45.3 | 47.8 | 46.7 | 48.3 |
| horse   | 22.2 | 32.3 | 21.9 | 34.1 | 21.9 | 24.7 | 21.9 | 12.6 | 24.7 | 21.1 |
| motorbike                                     | 33.7 | 40.1 | 31.2 | 40.1 | 31.2 | 35.4 | 31.2 | 38.3 | 34.2 | 32.7 |
| train   | 31.5 | 47.8 | 34.4 | 45.5 | 34.4 | 25.3 | 34.4 | 23.3 | 32.7 | 31.0 |
| avg   | 31.1 | 41.0 | 31.0 | 40.2 | 31.0 | 32.8 | 31.0 | 29.3 | 30.5 | 31.9 |
| performance gap                               | 9.9  | -    | 9.2  | -    | 1.8  | -    | -1.8 | -    | 1.4  | -    |

Table B.7: Per-class mAP results of the R-CNN detector when testing on ILSVRC IMG.

| mAP of the R-CNN detector when testing on VID |      |      |      |      |      |      |      |      |      |      |
|---|------|------|------|------|------|------|------|------|------|------|
| classname                                     | SLA  |      | AD   |      | IQ-G |      | IQ-M |      | AS   |      |
|   | VID  | IMG  | VID  | IMG  | VID  | IMG  | VID  | IMG  | VID  | IMG  |
| aeroplane                                     | 49.7 | 35.0 | 50.6 | 29.3 | 50.6 | 19.7 | 50.6 | 18.6 | 30.1 | 28.7 |
| bird  | 29.2 | 23.8 | 27.4 | 23.9 | 27.4 | 27.0 | 27.4 | 24.6 | 27.2 | 27.0 |
| boat  | 44.0 | 36.6 | 42.7 | 36.6 | 42.7 | 26.5 | 42.7 | 35.4 | 34.7 | 26.9 |
| car   | 37.4 | 35.1 | 36.7 | 34.0 | 36.7 | 31.7 | 36.7 | 33.2 | 34.7 | 32.0 |
| cat   | 12.8 | 15.9 | 13.6 | 17.9 | 13.6 | 20.3 | 13.6 | 14.8 | 14.2 | 22.2 |
| cow   | 16.8 | 25.4 | 10.2 | 23.0 | 10.2 | 23.4 | 10.2 | 9.70 | 17.5 | 26.8 |
| dog   | 29.0 | 23.0 | 29.1 | 21.1 | 29.1 | 26.9 | 29.1 | 28.3 | 32.2 | 29.2 |
| horse   | 36.6 | 17.2 | 36.9 | 13.5 | 36.9 | 30.3 | 36.9 | 14.3 | 37.0 | 29.8 |
| motorbike                                     | 44.7 | 36.7 | 46.1 | 37.6 | 46.1 | 39.4 | 46.1 | 48.0 | 41.5 | 38.9 |
| train   | 30.8 | 24.9 | 28.6 | 21.3 | 28.6 | 21.8 | 28.6 | 21.9 | 28.5 | 25.3 |
| avg   | 33.1 | 27.4 | 32.2 | 25.8 | 32.2 | 26.7 | 32.2 | 24.9 | 29.8 | 28.7 |
| performance gap                               | -    | 5.7  | -    | 6.4  | -    | 5.5  | -    | 7.2  | -    | 1.1  |

Table B.8: Per-class mAP results of the R-CNN detector when testing on ILSVRC VID.



# Appendix C

## Mining video training data

In this appendix, we explore the performance of object detectors, when trained from both still images and video frames. The goal is to add video training frames that improve the detection performance over training just with images. In particular, we aim at detecting image test samples that are not detected when training alone with images. Thus, we mine video training data from a large pool of videos that match the distribution of these undetected test samples.

**Motivation.** The Sections 3.4.1-3.4.6 show that the appearance diversity and the aspect distribution of a training set play a very important role for object detection. Given that the space of possible samples for an object class is very large, any given dataset invariably samples it in a limited way with its own specific bias [Torralba and Efros, 2011]. Inspired by the fact that for good performance it is important to collect a broad range of videos showing all aspects expected to appear in the test set, we train an object detector with both images and videos. We mine video training data from a large collection of videos so as to *cover* (i.e. help to detect) undetected image test samples.

For collecting a broad range of training videos, we use the VIRAT video dataset of Oh et al. [2011a]. It comprises 329 videos with bounding-box annotations for 5 moving object categories: person, car, vehicles, objects, and bike. We work only on one class: car. The VIRAT dataset has 215 videos of car with more than 6 million bounding-box annotations. To avoid redundant samples, instead of using all the car data, we sample 100k frames that contain approximately 107k different car samples. For images, we use the VOC dataset [Everingham et al., 2007], that, for the class car, contains 1.2k and 1.6k annotated training and test instances, respectively.

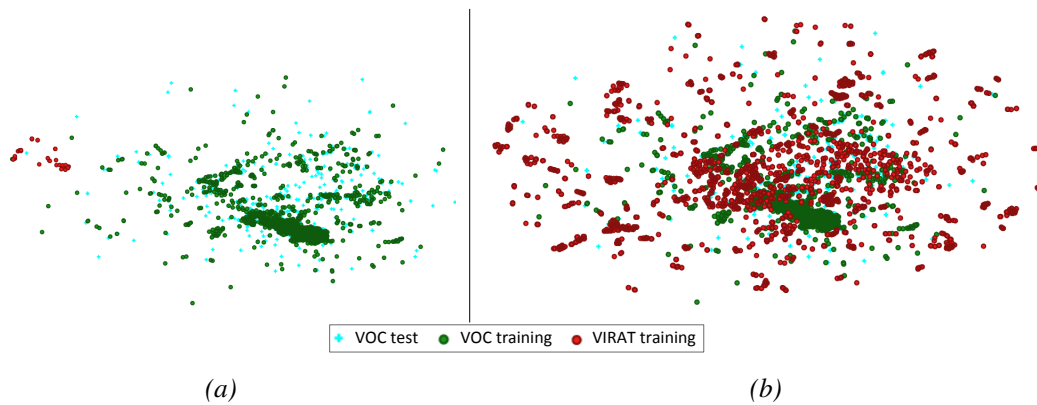


Figure C.1: 2D visualization of the training and test sets (a) before and (b) after selecting VIRAT samples. We select as many VIRAT samples as there are in the first 10 NNs of each missed detection and of each correct detection (testVOC).

**Implementation details.** We use the R-CNN [Girshick et al., 2014a] and the Fast R-CNN [Girshick, 2015a] detectors; see Sections 2.2.2-2.2.3 for more details. For object proposals we use Selective Search [Uijlings et al., 2013]. In all our experiments, we use AlexNet [Krizhevsky et al., 2012] as the underlying CNN architecture.

**Baselines.** We train the R-CNN [Girshick et al., 2014a] and the Fast R-CNN [Girshick, 2015a] detectors with trainVOC (including other classes) and we test on testVOC. The baseline mAPs for the class car are 59.6% and 63.7% for each detector, respectively. Then, we examine the detections, see first row of Figure C.2. For a given precision 1% the recall is 77.6%, indicating that most of the samples have been detected. This also shows that 22.4% of the samples still need to be detected.

**Mining video samples.** Our goal is to augment the images training set with video samples, so as to improve detection performance on images. In particular, we are interested in mining video samples (VIRAT) that detect previously missed detections. We work on 1% precision, that corresponds to 22.4% missed detections, i.e. 269 out of the 1.2k test samples are not detected.

We take into account the test samples: for each test detection (correct or missed), we compute its Euclidean distance from all VIRAT samples and from all VOC training samples. Then, we evaluate the four following strategies for mining VIRAT samples:

1. select a VIRAT sample if it is the first Nearest Neighbour (NN) of each missed detection and of each correct detection,

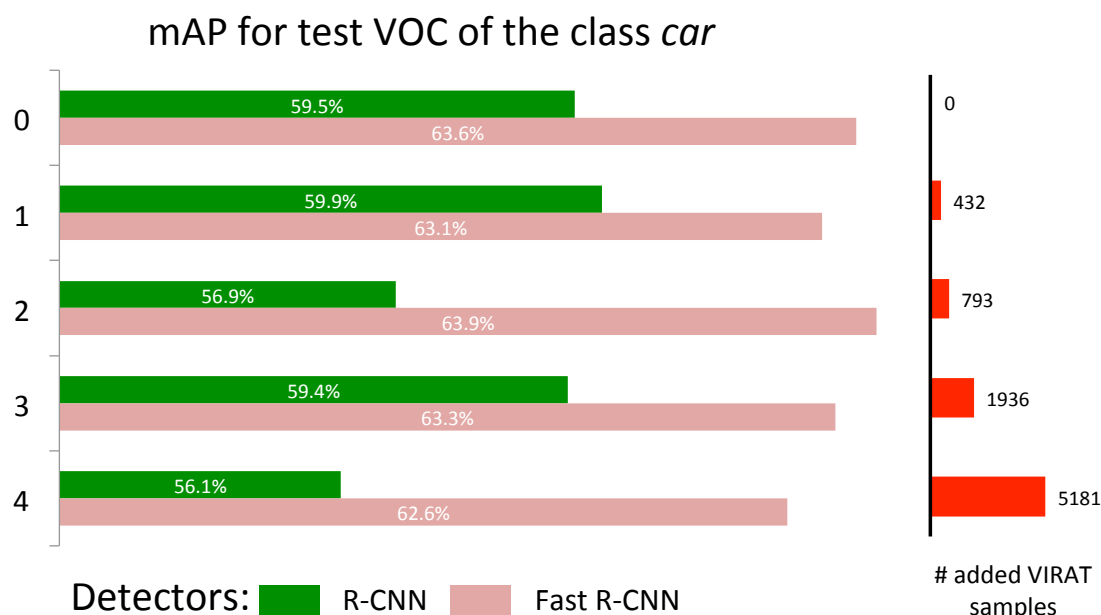


Figure C.2: % mAP for the class *car* of the R-CNN and fast R-CNN detectors for different training sets. Each training set contains the 1644 VOC training samples and some VIRAT samples.

2. select a VIRAT sample if it is the first or the second NN of each missed detection and of each correct detection,
3. select as many VIRAT samples as there are in the first 10 NNs of each missed detection,
4. select as many VIRAT samples as there are in the first 10 NNs of each missed detection and of each correct detection

Figure C.1 shows the selection process of VIRAT samples for the last strategy.

Then, we augment the VOC training set with the VIRAT samples coming from the four aforementioned strategies, and we train the two detectors. We report the mAP results in Figure C.2. We observe that by adding more video training data not only the detection performance is not increased, but it drops.

We also evaluate the detections before and after adding VIRAT samples (Figure C.3). We observe that the more video training samples we add, the more test VOC samples we detect. But we also observe that there is a significant amount of samples that were detected correctly before adding the video samples, but are no longer detected after. Therefore, we conclude that adding video training data steers the detector

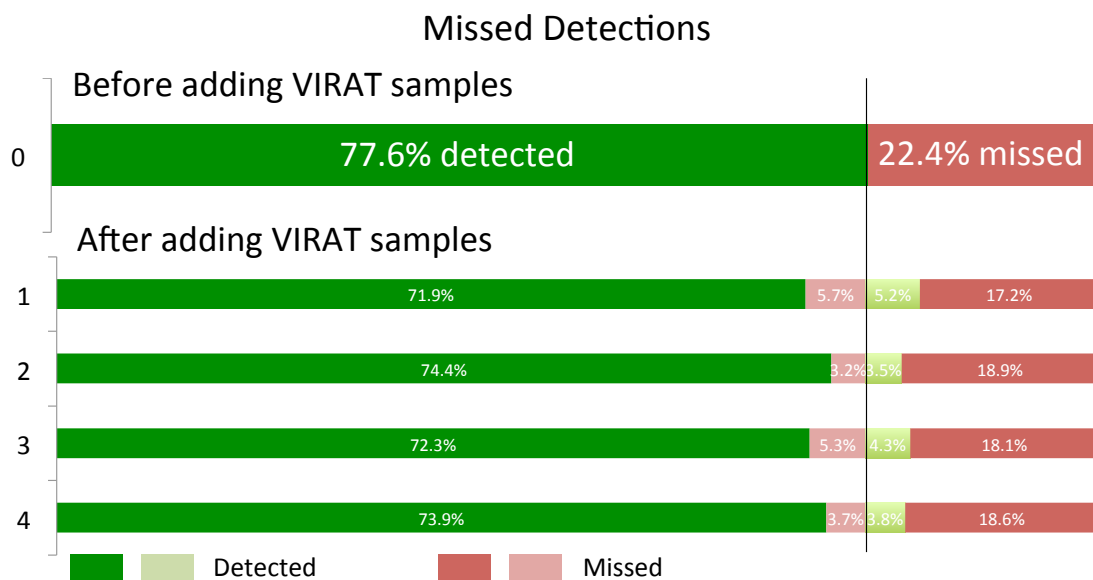


Figure C.3: % statistics of the detections before and after adding the VIRAT training samples for the class car.

away from the actual test data, and hence, we should consider a true domain adaptation method to use video training samples.



# Appendix D

## Additional experimental results on joint object and action detection

In this appendix we present the per-class results for all experiments of Chapter 4. First, in Tables D.1-D.3 we report the per-class results when examining the impact of the end-to-end fusion and of each modality (RGB and flow) and of the late and early fusion techniques (Table 4.3) for the A2D (Table D.1), YTO (Table D.2), and ILSVRC VID (Table D.3) datasets.

In Table 4.4 we examine the average detection performance of our multitask architecture and we compare it with the baseline and with other alternatives (Cartesian, hierarchical). Table D.4 and Table D.5 report the per-class mAP results when testing on objects or actions alone, while Table D.6 reports the per-class results when testing on objects and actions.

Table 4.6 reports the segmentation accuracy on A2D with class-average pixel accuracy (ave), global pixel accuracy (glo) and mean Intersection over Union (mIoU) metrics. In Table D.7 we report the per-class average pixel accuracy (ave) and global pixel accuracy (glo) for our multitask architecture and we compare it with the state of the art.

| mAP of A2D |      |      |             |             |
|------------|------|------|-------------|-------------|
| classname  | RGB  | Flow | late fusion | Our fusion  |
| adult      | 70.7 | 39.9 | 66.9        | 71.5        |
| baby       | 79.8 | 46.7 | 78.0        | 80.9        |
| ball       | 31.5 | 15.8 | 32.4        | 32.3        |
| bird       | 66.9 | 23.2 | 61.8        | 70.0        |
| car        | 67.4 | 44.6 | 69.5        | 71.4        |
| cat        | 58.8 | 23.3 | 57.0        | 61.3        |
| dog        | 66.4 | 29.9 | 65.3        | 69.5        |
| avg        | 63.1 | 31.9 | 61.6        | <b>65.3</b> |

Table D.1: Per-class mAP results on the objects of the A2D dataset while examining the end-to-end training.

| mAP of YTO |      |      |             |             |
|------------|------|------|-------------|-------------|
| classname  | RGB  | Flow | late fusion | Our fusion  |
| aeroplane  | 62.5 | 50.9 | 68.7        | 72.5        |
| bird       | 83.2 | 65.5 | 89.6        | 84.0        |
| boat       | 34.7 | 35.9 | 42.2        | 48.8        |
| car        | 57.9 | 36.4 | 56.2        | 59.2        |
| cat        | 60.8 | 14.7 | 52.8        | 59.1        |
| cow        | 40.9 | 12.6 | 34.8        | 42.4        |
| dog        | 68.8 | 16.5 | 59.0        | 73.6        |
| horse      | 46.3 | 15.3 | 42.6        | 47.5        |
| motorbike  | 65.5 | 40.8 | 62.4        | 69.2        |
| train      | 68.2 | 33.7 | 64.0        | 65.5        |
| avg        | 58.9 | 32.2 | 57.2        | <b>62.2</b> |

Table D.2: Per-class mAP results on the objects of the YTO dataset while examining the end-to-end training.

| mAP of VID   |      |      |             |             |
|--------------|------|------|-------------|-------------|
| classname    | RGB  | Flow | late fusion | Our fusion  |
| airplane     | 82.0 | 3.46 | 10.3        | 82.6        |
| antelope     | 61.1 | 0.21 | 47.0        | 60.8        |
| bear         | 40.3 | 1.41 | 11.2        | 52.6        |
| bicycle      | 48.5 | 11.6 | 49.6        | 55.2        |
| bird         | 41.6 | 0.30 | 39.7        | 45.2        |
| bus          | 58.5 | 0.32 | 38.9        | 63.1        |
| car          | 46.6 | 2.53 | 26.0        | 48.3        |
| cattle       | 27.4 | 6.57 | 31.2        | 32.8        |
| dog          | 36.9 | 6.74 | 33.9        | 44.2        |
| domestic cat | 40.4 | 0.39 | 14.3        | 49.4        |
| elephant     | 59.7 | 0.39 | 49.4        | 58.9        |
| fox          | 60.4 | 4.92 | 58.3        | 61.7        |
| giant panda  | 62.0 | 8.42 | 61.8        | 68.0        |
| hamster      | 34.9 | 17.4 | 41.3        | 52.0        |
| horse        | 52.8 | 28.5 | 59.6        | 57.9        |
| lion         | 1.74 | 0.15 | 2.01        | 2.54        |
| lizard       | 27.7 | 0.08 | 9.43        | 21.6        |
| monkey       | 25.1 | 0.48 | 28.3        | 25.6        |
| motorcycle   | 68.9 | 20.9 | 65.8        | 74.8        |
| rabbit       | 23.5 | 6.65 | 24.2        | 25.8        |
| red panda    | 18.9 | 0.08 | 20.2        | 12.8        |
| sheep        | 49.2 | 0.06 | 41.7        | 48.1        |
| snake        | 2.66 | 0.21 | 5.27        | 5.10        |
| squirrel     | 11.2 | 0.21 | 10.4        | 15.3        |
| tiger        | 73.6 | 0.44 | 64.1        | 78.8        |
| train        | 64.4 | 22.8 | 58.8        | 68.5        |
| turtle       | 50.4 | 0.38 | 3.95        | 42.1        |
| watercraft   | 52.3 | 0.84 | 32.4        | 51.7        |
| whale        | 52.9 | 1.06 | 48.9        | 55.7        |
| zebra        | 79.2 | 0.02 | 27.1        | 81.6        |
| avg          | 45.2 | 4.92 | 33.8        | <b>48.1</b> |

Table D.3: *Per-class mAP results on the objects of the ILSVRC 2015 VID dataset while examining the end-to-end training.*

| mAP when testing only on the objects of A2D |         |         |          |           |              |             |
|---|---------|---------|----------|-----------|--------------|-------------|
| classname                                   | objects | actions | baseline | Cartesian | hierarchical | multitask   |
| adult                                       | 71.5    | -       | -        | 71.9      | 72.8         | 74.2        |
| baby  | 80.9    | -       | -        | 85.3      | 86.1         | 85.0        |
| ball  | 32.3    | -       | -        | 36.6      | 32.4         | 36.8        |
| bird  | 70.0    | -       | -        | 72.9      | 73.5         | 73.9        |
| car   | 71.4    | -       | -        | 70.2      | 75.1         | 73.6        |
| cat   | 61.3    | -       | -        | 63.3      | 63.0         | 64.1        |
| dog   | 69.5    | -       | -        | 70.6      | 72.1         | 70.7        |
| avg   | 65.3    | -       | -        | 67.2      | 67.9         | <b>68.3</b> |

Table D.4: Per-class mAP results for different architectures when testing only on the objects of the A2D dataset.

| mAP when testing only on the actions of A2D |         |         |          |             |              |           |
|---|---------|---------|----------|-------------|--------------|-----------|
| classname                                   | objects | actions | baseline | Cartesian   | hierarchical | multitask |
| climbing                                    | -       | 63.0    | -        | 68.8        | 66.1         | 63.4      |
| crawling                                    | -       | 64.4    | -        | 66.5        | 63.9         | 68.5      |
| eating                                      | -       | 67.3    | -        | 69.5        | 68.2         | 70.3      |
| flying                                      | -       | 49.6    | -        | 52.1        | 51.5         | 52.0      |
| jumping                                     | -       | 42.6    | -        | 47.0        | 47.0         | 47.2      |
| rolling                                     | -       | 60.0    | -        | 63.0        | 64.1         | 62.1      |
| running                                     | -       | 46.9    | -        | 51.1        | 52.6         | 52.4      |
| walking                                     | -       | 56.0    | -        | 63.8        | 63.2         | 63.9      |
| avg   | -       | 56.2    | -        | <b>60.2</b> | 59.6         | 60.0      |

Table D.5: Per-class mAP results for different architectures when testing only on the actions of the A2D dataset.

| mAP when testing both on the objects and the actions of A2D |             |             |             |             |             |             |             |             |             |             |             |             |
|---|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
|   | adult       |             |             |             |             |             |             |             | ball        |             |             |             |
|   | climbing    | crawling    | eating      | jumping     | rolling     | running     | walking     | none        | flying      | jumping     | rolling     | none        |
| Baseline  | 62.1        | 67.1        | 74.6        | 44.5        | 45.4        | 48.1        | 46.9        | 20.4        | 11.1        | 23.4        | 39.3        | 4.3         |
| Cartesian   | <b>79.2</b> | 70.2        | <b>86.1</b> | 44.2        | 55.6        | 53.3        | <b>60.8</b> | 45.8        | <b>23.1</b> | 32.2        | 41.4        | <b>21.7</b> |
| Hierarchical  | 77.7        | <b>70.6</b> | 85.3        | 45.0        | <b>58.4</b> | 52.4        | 56.2        | 45.4        | 11.4        | <b>35.5</b> | <b>43.3</b> | 11.6        |
| Multitask   | 74.7        | 70.5        | 78.7        | <b>45.5</b> | 49.6        | <b>58.0</b> | 57.3        | <b>46.3</b> | 19.2        | 30.3        | 41.8        | 20.5        |
|   | baby        |             |             |             |             | bird        |             |             |             |             |             |             |
|   | climbing    | crawling    | rolling     | walking     | none        | climbing    | eating      | flying      | jumping     | rolling     | walking     | none        |
| Baseline  | 63.4        | 66.1        | 79.1        | 71.3        | 5.9         | 50.9        | <b>44.2</b> | 66.1        | 26.1        | 51.7        | 50.2        | 2.1         |
| Cartesian   | 66.4        | 70.4        | 81.5        | 76.7        | 31.3        | <b>56.0</b> | 34.7        | <b>67.9</b> | 28.8        | 57.0        | 51.9        | 3.4         |
| Hierarchical  | <b>69.1</b> | 69.7        | 80.0        | 74.4        | <b>36.0</b> | 51.3        | 39.3        | 67.5        | 27.1        | <b>58.4</b> | 52.7        | <b>4.5</b>  |
| Multitask   | 63.1        | <b>76.9</b> | <b>82.8</b> | <b>77.2</b> | 24.0        | 49.4        | 41.7        | 67.8        | <b>30.4</b> | 55.5        | <b>54.3</b> | 3.8         |
|   | car         |             |             |             |             | cat         |             |             |             |             |             |             |
|   | flying      | jumping     | rolling     | running     | none        | climbing    | eating      | jumping     | rolling     | running     | walking     | none        |
| Baseline  | 43.9        | 86.4        | 64.6        | 62.1        | 3.8         | 63.0        | 0.0         | 20.8        | 55.2        | 27.3        | 49.2        | 2.1         |
| Cartesian   | 45.1        | <b>97.4</b> | 66.8        | 64.7        | 11.5        | 65.0        | 53.5        | 26.6        | 62.9        | 25.0        | 52.6        | 8.8         |
| Hierarchical  | <b>49.3</b> | 90.0        | <b>69.7</b> | <b>68.4</b> | <b>17.8</b> | <b>69.2</b> | 53.7        | <b>30.6</b> | <b>63.1</b> | <b>31.7</b> | <b>55.8</b> | 9.0         |
| Multitask   | 42.2        | 94.7        | 63.6        | 63.0        | 15.5        | 66.4        | <b>58.2</b> | 26.4        | 62.0        | 27.0        | 50.9        | <b>14.6</b> |
|   | dog         |             |             |             |             |             |             |             | avg         |             |             |             |
|   | crawling    | eating      | jumping     | rolling     | running     | walking     | none        |             |             |             |             |             |
| Baseline  | 44.4        | 65.9        | 31.9        | 28.4        | 29.5        | 58.3        | 1.1         | 43.1        |             |             |             |             |
| Cartesian   | <b>54.4</b> | <b>73.2</b> | <b>38.3</b> | 35.9        | 30.1        | 59.9        | 2.6         | 49.2        |             |             |             |             |
| Hierarchical  | 48.4        | 72.9        | 34.4        | 34.5        | <b>33.3</b> | <b>62.8</b> | <b>13.9</b> | <b>49.6</b> |             |             |             |             |
| Multitask   | 48.7        | 72.9        | 36.5        | <b>37.8</b> | 32.1        | 62.1        | 5.4         | 48.9        |             |             |             |             |

Table D.6: *Per-class mAP results for different architectures when testing both on the objects and on their actions for the A2D dataset.*

| Segmentation accuracy for objects and actions on A2D |             |             |             |             |             |             |             |             |             |             |             |             |
|--|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
|  | adult       |             |             |             |             |             |             |             | ball        |             |             |             |
|  | climbing    | crawling    | eating      | jumping     | rolling     | running     | walking     | none        | flying      | jumping     | rolling     | none        |
| Trilayer [Xu et al., 2015]                           | 33.1        | 59.8        | 49.8        | 19.9        | 27.6        | 40.2        | 31.7        | 24.6        | 1           | 11.9        | 6.1         | 0.0         |
| GPM (TSP) [Xu and Corso, 2016]                       | 74.6        | <b>79.8</b> | 70.7        | <b>49.3</b> | 51.5        | 50.6        | 40.4        | 0.0         | 11.3        | 27.0        | 20.8        | 0.0         |
| GPM (GBH) [Xu and Corso, 2016]                       | <b>74.8</b> | 81.0        | 76.4        | <b>49.3</b> | <b>52.4</b> | 50.4        | 41.0        | 0.0         | 11.3        | <b>28.3</b> | 21.1        | 0.0         |
| <b>Ours (GBH)</b>                                    | 59.1        | 61.2        | <b>81.6</b> | 47.3        | 43.8        | <b>52.6</b> | <b>50.9</b> | <b>60.7</b> | <b>13.8</b> | 10.3        | 73.8        | 48.6        |
| <b>Ours (SharpMask)</b>                              | 60.4        | 66.6        | 77.8        | 44.5        | 48.1        | 47.6        | 47.0        | 57.2        | 6.0         | 12.7        | <b>81.2</b> | <b>52.5</b> |
|  | baby        |             |             |             |             | bird        |             |             |             |             |             |             |
|  | climbing    | crawling    | rolling     | walking     | none        | climbing    | eating      | flying      | jumping     | rolling     | walking     | none        |
| Trilayer [Xu et al., 2015]                           | 20.4        | 21.7        | 39.3        | 25.3        | 0.0         | 28.1        | 18.2        | 55.3        | 20.3        | 42.5        | 9.0         | 0.0         |
| GPM (TSP) [Xu and Corso, 2016]                       | 65.3        | 64.7        | 57.2        | 60.5        | 0.0         | <b>62.2</b> | 37.1        | 66.6        | 17.4        | 45.4        | 42.2        | 0.0         |
| GPM (GBH) [Xu and Corso, 2016]                       | <b>65.4</b> | <b>65.0</b> | 58.4        | <b>61.5</b> | 0.0         | 60.6        | 38.8        | 66.5        | 17.5        | <b>45.9</b> | 47.9        | 0.0         |
| <b>Ours (GBH)</b>                                    | 62.3        | 57.7        | <b>77.6</b> | 53.3        | <b>19.7</b> | 54.9        | 40.4        | <b>74.3</b> | 25.9        | 39.2        | 49.3        | <b>33.9</b> |
| <b>Ours (SharpMask)</b>                              | 51.1        | 57.1        | 72.2        | 47.1        | 17.6        | 52.6        | <b>42.0</b> | 57.5        | <b>26.6</b> | 44.9        | <b>50.5</b> | 33.2        |
|  | car         |             |             |             |             | cat         |             |             |             |             |             |             |
|  | flying      | jumping     | rolling     | running     | none        | climbing    | eating      | jumping     | rolling     | running     | walking     | none        |
| Trilayer [Xu et al., 2015]                           | 24.4        | 75.9        | 44.3        | 48.3        | 2.4         | 33.1        | 27.2        | 6.1         | 49.8        | <b>48.5</b> | 6.6         | 0.0         |
| GPM (TSP) [Xu and Corso, 2016]                       | <b>42.9</b> | 84.5        | 69.2        | 64.8        | 0.0         | 42.5        | 49.3        | 31.9        | 71.1        | 46.4        | 18.8        | 0.0         |
| GPM (GBH) [Xu and Corso, 2016]                       | 41.2        | 86.3        | <b>70.9</b> | <b>65.9</b> | 0.0         | 42.8        | 52.3        | <b>33.7</b> | <b>71.7</b> | 48.0        | 19.1        | 0.0         |
| <b>Ours (GBH)</b>                                    | 36.4        | 83.3        | 57.6        | 56.9        | 46.6        | <b>62.4</b> | <b>62.2</b> | 15.9        | 68.7        | 35.7        | <b>35.4</b> | <b>10.0</b> |
| <b>Ours (SharpMask)</b>                              | 37.2        | <b>91.4</b> | 57.3        | 54.3        | <b>48.9</b> | 60.7        | 61.1        | 15.2        | 68.8        | 33.9        | 33.7        | 7.76        |
|  | dog         |             |             |             |             |             |             |             | Background  | avg         | glo         |             |
|  | crawling    | eating      | jumping     | rolling     | running     | walking     | none        | none        |             |             |             |             |
| Trilayer [Xu et al., 2015]                           | 9.9         | 31          | 2.0         | 27.6        | 23.6        | 39.4        | 0.0         | 78.5        | 26.5        | 72.9        |             |             |
| GPM (TSP) [Xu and Corso, 2016]                       | <b>45.3</b> | 60.2        | 31.3        | 62.5        | <b>25.8</b> | 74.0        | 0.0         | 89.1        | 43.3        | 84.2        |             |             |
| GPM (GBH) [Xu and Corso, 2016]                       | 44.1        | 61.5        | <b>31.4</b> | <b>62.6</b> | 25.7        | <b>74.2</b> | 0.0         | 88.4        | 43.9        | 83.8        |             |             |
| <b>Ours (GBH)</b>                                    | 30.6        | 56.6        | 27.1        | 55.0        | 14.5        | 70.5        | 4.3         | 87.5        | <b>48.0</b> | 83.9        |             |             |
| <b>Ours (SharpMask)</b>                              | 31.5        | <b>62.1</b> | 26.4        | 59.5        | 14.0        | 68.2        | <b>5.7</b>  | <b>93.1</b> | 47.5        | <b>88.7</b> |             |             |

Table D.7: Comparison of per-class segmentation results (ave and glo) when using object-action labels on the A2D dataset.

# Appendix E

## ACT-detector: additional experimental results

In this appendix we present the per-class results of Chapter 5 for the three action localization datasets: UCF-Sports, J-HMDB, and UCF-101. For J-HMDB we report results on split 1, unless stated otherwise.

In Tables-E.1-E.3, we present all the per-class results that are related with our ACT-detector, i.e. per-frame results. In particular, we present the MABO and classification accuracy for  $K = 1$  and  $K = 6$  (Figure 5.10), the frame-mAP results for fast moving objects (Table 5.1), and the frame-mAP (Table 5.3).

In Tables-E.4-E.6, we present all the per-class results that are related with our tubes. In particular, we present the MABO for  $K = 1$  and  $K = 6$  (Figure 5.13), and the video-mAP for  $K = 1$  at threshold  $\delta = 0.5$  and for  $K = 6$  at various thresholds  $\delta = 0.2, 0.5, 0.75$  (Table 5.4). Finally, in Table E.7 we present frame-mAP and video-mAP per-class results for the splits 2 and 3 of J-HMDB.

| UCF-Sports     |      |      |              |      |                     |      |           |      |
|----------------|------|------|--------------|------|---------------------|------|-----------|------|
| classname      | MABO |      | Cls accuracy |      | frame-mAP fast Obj. |      | frame-mAP |      |
|                | K=1  | K=6  | K=1          | K=6  | K=1                 | K=6  | K=1       | K=6  |
| diving         | 80.3 | 84.9 | 100          | 100  | 100                 | 99.4 | 99.9      | 99.4 |
| golf           | 84.8 | 86.4 | 83.3         | 85   | 31.2                | 100  | 86.4      | 89.3 |
| kicking        | 82.4 | 85.5 | 52.8         | 74.6 | 45.9                | 83.1 | 36.9      | 67.5 |
| lifiting       | 91.7 | 93.6 | 100          | 100  | 100                 | 100  | 100       | 100  |
| riding         | 77.8 | 82.9 | 95.4         | 98.7 | 100                 | 100  | 100       | 100  |
| run            | 74.9 | 85.4 | 80.7         | 96.1 | 73.6                | 89.5 | 72.2      | 88.7 |
| skate boarding | 76.2 | 81.5 | 6.78         | 12.8 | 48.9                | 62   | 55.6      | 65.4 |
| swing1         | 78.9 | 85.0 | 54           | 79   | 96.9                | 98.1 | 97.1      | 98.4 |
| swing2         | 81.2 | 83.8 | 97.6         | 99.3 | 100                 | 99.5 | 100       | 99.5 |
| walk           | 70.8 | 80.5 | 75.3         | 88.0 | 42.0                | 47.4 | 58.0      | 68.4 |
| avg            | 79.9 | 84.9 | 74.6         | 83.3 | 73.8                | 87.9 | 80.6      | 87.6 |

Table E.1: Per-class results for frame-related evaluations on UCF-Sports.



| J-HMDB (split 1) |      |      |              |      |                     |      |           |      |
|------------------|------|------|--------------|------|---------------------|------|-----------|------|
| classname        | MABO |      | Cls accuracy |      | frame-mAP fast Obj. |      | frame-mAP |      |
|                  | K=1  | K=6  | K=1          | K=6  | K=1                 | K=6  | K=1       | K=6  |
| brush hair       | 87.9 | 88.2 | 78.9         | 81.8 | 100                 | 100  | 88.6      | 92   |
| catch            | 86.0 | 88.2 | 19.2         | 48.3 | 14.3                | 29   | 17.5      | 41.2 |
| clap             | 82.5 | 85.9 | 70.4         | 55.1 | 100                 | 100  | 70.3      | 66.5 |
| climb stairs     | 83.8 | 85.6 | 38.8         | 75.2 | 44.1                | 54.1 | 67.9      | 81.6 |
| golf             | 88.4 | 88.8 | 99.3         | 99.7 | 93.3                | 95.5 | 95.1      | 94.9 |
| jump             | 67.5 | 75.5 | 33.0         | 34.2 | 34                  | 34.8 | 22.9      | 26.7 |
| kick ball        | 78.8 | 81.1 | 37.3         | 44.1 | 37.4                | 51.2 | 25.3      | 38.3 |
| pick             | 81.0 | 83.1 | 43.5         | 58.2 | 65.0                | 77.8 | 50.9      | 72.9 |
| pour             | 87   | 89.7 | 76.8         | 89.7 | 61.5                | 79.9 | 85.7      | 90.3 |
| pullup           | 86.9 | 87.1 | 90.4         | 98.9 | 72.7                | 98.8 | 89.8      | 99.2 |
| push             | 76.4 | 78.5 | 67.3         | 74.3 | 57.9                | 58.8 | 76.4      | 76.8 |
| run              | 79.3 | 83.6 | 47.4         | 46.0 | 47.1                | 50.4 | 49.1      | 49.0 |
| shoot ball       | 80.9 | 81.9 | 22.3         | 29.8 | 40.9                | 37.4 | 29.4      | 30.5 |
| shoot bow        | 86.8 | 88.0 | 72.9         | 88.8 | 58.9                | 100  | 94.3      | 96.6 |
| shoot gun        | 90.6 | 91.7 | 30.9         | 52.7 | 12.1                | 50.5 | 52.3      | 53.4 |
| sit              | 81.8 | 87.2 | 55.1         | 62.9 | 48.5                | 65.2 | 49.8      | 67.4 |
| stand            | 84.0 | 87.7 | 53.6         | 62.2 | 28.0                | 21.4 | 60.8      | 71.1 |
| swing baseball   | 84.0 | 84.3 | 32.2         | 7.12 | 35.9                | 12.5 | 79.2      | 54.0 |
| throw            | 81.1 | 82.8 | 31.2         | 35.1 | 41.8                | 38.6 | 34.9      | 35.0 |
| walk             | 84.4 | 88.6 | 52.6         | 68.7 | 44.9                | 54.5 | 56.5      | 66   |
| wave             | 84.1 | 83.6 | 39.4         | 45.8 | 1.48                | 0    | 45.5      | 48.2 |
| avg              | 83.0 | 85.3 | 52.0         | 59.9 | 49.5                | 57.6 | 59.1      | 64.4 |

Table E.2: Per-class results for frame-related evaluations on J-HMDB.

| UCF-101            |      |      |              |      |                     |      |           |      |
|--------------------|------|------|--------------|------|---------------------|------|-----------|------|
| classname          | MABO |      | Cls accuracy |      | frame-mAP fast Obj. |      | frame-mAP |      |
|                    | K=1  | K=6  | K=1          | K=6  | K=1                 | K=6  | K=1       | K=6  |
| basketball         | 79.7 | 82.9 | 77.9         | 77.6 | 58.1                | 70.0 | 31.5      | 49.4 |
| basketball dunk    | 70.2 | 75.2 | 74.5         | 87.3 | 56.5                | 50.1 | 48.3      | 44.1 |
| biking             | 77.5 | 82.9 | 70.3         | 85.4 | 70.5                | 71.5 | 62.8      | 64.5 |
| cliff diving       | 77.0 | 82.6 | 80.5         | 88.2 | 75.4                | 78.1 | 63.6      | 69.7 |
| cricket bowling    | 78.3 | 84.3 | 54.3         | 52.5 | 73.1                | 69.1 | 43.7      | 39.9 |
| diving             | 74.9 | 80.7 | 87.7         | 95.8 | 89.1                | 88.1 | 76.9      | 76.7 |
| fencing            | 83.1 | 87.3 | 93.4         | 94.1 | 88.8                | 89.2 | 87.6      | 88.1 |
| floor gymnastics   | 80.5 | 84.6 | 84.3         | 90.1 | 89.7                | 90.9 | 86.8      | 89.1 |
| golf swing         | 80.9 | 86.6 | 76.7         | 81.4 | 6.02                | 8.05 | 52.1      | 56.1 |
| horse riding       | 75.2 | 78.8 | 90.3         | 95.5 | 92.1                | 92.1 | 91.5      | 92.1 |
| ice dancing        | 79.1 | 83.2 | 97.2         | 98.1 | 62.7                | 63.3 | 63.8      | 63.6 |
| long jump          | 73.9 | 80.1 | 61.7         | 67.6 | 72.3                | 71.8 | 67.8      | 66.8 |
| pole vault         | 75.8 | 81.9 | 67.2         | 73.7 | 71.9                | 72.5 | 64.3      | 64.5 |
| rope climbing      | 77.1 | 83.8 | 77.2         | 91.7 | 88.9                | 95.8 | 86.2      | 91.2 |
| salsa spin         | 73.8 | 78.7 | 77.2         | 82.6 | 58.7                | 61.2 | 33.4      | 34.4 |
| skate boarding     | 81.6 | 85.2 | 79.5         | 83.0 | 84.7                | 86.3 | 82.0      | 83.8 |
| skiing             | 80.0 | 83.6 | 67.5         | 70.5 | 72.2                | 71.5 | 73.3      | 72.4 |
| skijet             | 75.9 | 79.8 | 86.1         | 91.7 | 85.3                | 80.3 | 85.7      | 80.8 |
| soccer juggling    | 77.9 | 83.0 | 77.3         | 92.4 | 80.5                | 84.2 | 86.1      | 91.3 |
| surfing            | 78.4 | 84.3 | 87.7         | 91.6 | 92.3                | 89.1 | 76.6      | 72.8 |
| tennis swing       | 74.2 | 80.7 | 49.4         | 52.0 | 62.9                | 50.4 | 27.1      | 24.7 |
| trampoline jumping | 73.4 | 79.4 | 78.0         | 90.7 | 67.3                | 70.0 | 55.4      | 56.5 |
| volleyball spiking | 73.3 | 78.7 | 63.8         | 65.0 | 57.0                | 51.1 | 26.9      | 24.2 |
| walking with dog   | 79.3 | 83.9 | 72.1         | 83.3 | 81.9                | 80.5 | 78.6      | 79.1 |
| avg                | 77.1 | 82.2 | 76.3         | 82.6 | 72.4                | 72.3 | 64.7      | 65.7 |

Table E.3: Per-class results for frame-related evaluations on UCF-101.

| UCF-Sports     |            |      |                |                |                |                 |
|----------------|------------|------|----------------|----------------|----------------|-----------------|
| classname      | video-MABO |      | video-mAP      |                |                |                 |
|                | K=1        | K=6  | K=1            | K=6            |                |                 |
|                |            |      | $\delta = 0.5$ | $\delta = 0.2$ | $\delta = 0.5$ | $\delta = 0.75$ |
| diving         | 79.9       | 80.4 | 100            | 100            | 100            | 100             |
| golf           | 82.9       | 82.3 | 97.4           | 100            | 100            | 100             |
| kicking        | 77.2       | 68.9 | 75.7           | 83.3           | 83.3           | 62.9            |
| lifting        | 91.7       | 91.9 | 100            | 100            | 100            | 100             |
| riding         | 77.4       | 77.1 | 100            | 100            | 100            | 67.7            |
| run            | 61         | 81.1 | 73.7           | 90.1           | 90.1           | 90.1            |
| skate boarding | 71.8       | 73.0 | 75.2           | 73.3           | 73.3           | 39.4            |
| swing1         | 74.8       | 78.0 | 100            | 100            | 100            | 83.3            |
| swing2         | 80.9       | 79.6 | 100            | 100            | 100            | 100             |
| walk           | 70.5       | 74.3 | 72.2           | 80.3           | 80.3           | 40.5            |
| avg            | 76.8       | 78.7 | 89.4           | 92.7           | 92.7           | 78.4            |

Table E.4: *Per-class results for video-related evaluations on UCF-Sports.*

| J-HMDB (split 1) |            |      |                |                |                |                 |
|------------------|------------|------|----------------|----------------|----------------|-----------------|
| classname        | video-MABO |      | video-mAP      |                |                |                 |
|                  | K=1        | K=6  | K=1            | K=6            |                |                 |
|                  |            |      | $\delta = 0.5$ | $\delta = 0.2$ | $\delta = 0.5$ | $\delta = 0.75$ |
| brush hair       | 85.7       | 81.7 | 96.3           | 97.4           | 97.4           | 72.3            |
| catch            | 69.3       | 83.2 | 23.4           | 55.7           | 55.7           | 50.0            |
| clap             | 73.7       | 78.0 | 78.7           | 72.2           | 72.2           | 42.1            |
| climb stairs     | 80.3       | 78.2 | 81.9           | 92.0           | 92.0           | 58.3            |
| golf             | 87.7       | 84.3 | 99.3           | 98.6           | 98.6           | 98.6            |
| jump             | 48.8       | 60.6 | 38.7           | 45.4           | 45.4           | 0               |
| kick ball        | 71.8       | 71.2 | 41.2           | 43.5           | 43.5           | 5.19            |
| pick             | 74.4       | 75.2 | 68.9           | 87.9           | 87.9           | 25.8            |
| pour             | 78.1       | 82.6 | 91.2           | 94.3           | 94.3           | 94.3            |
| pullup           | 85.2       | 81.1 | 94.1           | 100            | 100            | 100             |
| push             | 74.7       | 70.2 | 86.4           | 89.9           | 86.6           | 19.7            |
| run              | 71.0       | 69.4 | 57.4           | 49.4           | 43.8           | 40.3            |
| shoot ball       | 79.0       | 69.5 | 33.2           | 28.4           | 28.4           | 8.05            |
| shoot bow        | 83.6       | 82.3 | 99.1           | 100            | 100            | 88.2            |
| shoot gun        | 66.4       | 80.0 | 65.5           | 62.3           | 58.5           | 56.8            |
| sit              | 78.3       | 77.3 | 55.8           | 81.5           | 81.5           | 56.6            |
| stand            | 79.5       | 78.0 | 71.7           | 87.1           | 87.1           | 34.7            |
| swing baseball   | 81.8       | 78.5 | 84.5           | 59.9           | 59.9           | 57.3            |
| throw            | 59.7       | 61.1 | 42.7           | 44.6           | 44.6           | 27.3            |
| walk             | 81.2       | 82.6 | 59.6           | 74.2           | 74.2           | 68.3            |
| wave             | 60.6       | 69.6 | 48.5           | 47.3           | 47.3           | 27.5            |
| avg              | 74.8       | 75.9 | 67.5           | 72.0           | 71.4           | 49.1            |

Table E.5: Per-class results for video-related evaluations on J-HMDB.

| UCF-101            |            |      |                |                |                |                 |
|--------------------|------------|------|----------------|----------------|----------------|-----------------|
| classname          | video-MABO |      | video-mAP      |                |                |                 |
|                    | K=1        | K=6  | K=1            | K=6            |                |                 |
|                    |            |      | $\delta = 0.5$ | $\delta = 0.2$ | $\delta = 0.5$ | $\delta = 0.75$ |
| basketball         | 26.2       | 26.8 | 0              | 48.2           | 0              | 0               |
| basketball dunk    | 33.2       | 34.0 | 3.73           | 80.7           | 5.47           | 0               |
| biking             | 62.9       | 63.1 | 55.2           | 80.3           | 58.1           | 36.7            |
| cliff diving       | 45.8       | 48.9 | 41.1           | 90.5           | 46.8           | 0               |
| cricket bowling    | 36.2       | 36.8 | 1.8            | 38.9           | 1.61           | 0               |
| diving             | 53.0       | 53.7 | 45.6           | 89.1           | 41.3           | 0.07            |
| fencing            | 73.2       | 72.9 | 74.6           | 93.4           | 80.1           | 65.0            |
| floor gymnastics   | 75.9       | 74.8 | 97.9           | 98.9           | 95.6           | 49.0            |
| golf swing         | 56.9       | 59.4 | 44.4           | 76.8           | 52.7           | 10.8            |
| horse riding       | 71.1       | 71.9 | 94.7           | 96.6           | 94.5           | 43.9            |
| ice dancing        | 60.9       | 61.4 | 53.9           | 70.2           | 55.7           | 5.58            |
| long jump          | 58.5       | 61.2 | 74.3           | 88.4           | 68.6           | 24.2            |
| pole vault         | 56.6       | 58.9 | 56.0           | 88.7           | 56.1           | 0.05            |
| rope climbing      | 68.1       | 70.6 | 91.9           | 91.8           | 91.8           | 30.3            |
| salsa spin         | 26.5       | 30.8 | 5.6            | 30.2           | 6.75           | 0.4             |
| skate boarding     | 76.6       | 76.5 | 90.2           | 94.3           | 90.4           | 59.9            |
| skiing             | 71.6       | 70.3 | 81.7           | 84.9           | 78.7           | 51.2            |
| skijet             | 67.7       | 63.0 | 86.0           | 92.2           | 77.3           | 19.9            |
| soccer juggling    | 71.7       | 73.0 | 89.5           | 91.9           | 91.9           | 68.4            |
| surfing            | 55.9       | 55.7 | 57.3           | 69.1           | 52.2           | 29.3            |
| tennis swing       | 27.5       | 29.2 | 0.14           | 49.4           | 0.67           | 0               |
| trampoline jumping | 43.7       | 45.3 | 20.3           | 71.8           | 26.8           | 4.43            |
| volleyball spiking | 20.1       | 22.3 | 0              | 23.0           | 0              | 0               |
| walking with dog   | 63.9       | 64.1 | 63.9           | 78.2           | 62.5           | 39.4            |
| avg                | 54.3       | 55.2 | 51.2           | 75.7           | 51.5           | 22.4            |

Table E.6: Per-class results for video-related evaluations on UCF-101.

| J-HMDB         |           |         |                |                |                 |                |                |                 |
|----------------|-----------|---------|----------------|----------------|-----------------|----------------|----------------|-----------------|
| classname      | frame-mAP |         | video-mAP      |                |                 |                |                |                 |
|                | split 2   | split 3 | split 2        |                |                 | split 3        |                |                 |
|                |           |         | $\delta = 0.2$ | $\delta = 0.5$ | $\delta = 0.75$ | $\delta = 0.2$ | $\delta = 0.5$ | $\delta = 0.75$ |
| brush hair     | 91.6      | 69.8    | 98.2           | 98.2           | 86.7            | 73.6           | 73.6           | 39.1            |
| catch          | 17.8      | 71.7    | 18.3           | 18.3           | 14.2            | 69.7           | 69.7           | 63.6            |
| clap           | 80.2      | 72.0    | 91.6           | 85.6           | 82.0            | 78.1           | 78.1           | 72.2            |
| climb stairs   | 81.7      | 69.9    | 92.4           | 82.4           | 23.5            | 74.8           | 74.8           | 60.3            |
| golf           | 93.0      | 89.9    | 95.0           | 95.0           | 95.0            | 89.8           | 89.8           | 89.8            |
| jump           | 40.7      | 15.5    | 56.2           | 42.6           | 0.11            | 23.9           | 23.9           | 11.7            |
| kick ball      | 68.5      | 65.8    | 86.5           | 86.5           | 9.71            | 88.3           | 88.3           | 27.0            |
| pick           | 42.8      | 55.4    | 51.9           | 51.9           | 25.0            | 67.3           | 67.3           | 34.2            |
| pour           | 97.2      | 98.7    | 99.6           | 99.6           | 99.6            | 100            | 100            | 82.1            |
| pullup         | 97.5      | 83.5    | 100            | 100            | 71.7            | 88.2           | 88.2           | 80.5            |
| push           | 88.4      | 80.6    | 99.3           | 99.3           | 19.8            | 84.6           | 84.6           | 70.6            |
| run            | 44.1      | 35.7    | 54.4           | 54.4           | 27.2            | 60.3           | 60.3           | 40.1            |
| shoot ball     | 82.5      | 31.1    | 88.7           | 88.7           | 62.2            | 46.0           | 46.0           | 30.9            |
| shoot bow      | 99.4      | 85.1    | 100            | 100            | 100             | 86.6           | 86.6           | 86.6            |
| shoot gun      | 87.4      | 63.6    | 94.8           | 94.8           | 87.0            | 88.9           | 88.9           | 60.7            |
| sit            | 43.3      | 48.4    | 58.6           | 58.6           | 27.6            | 66.4           | 66.4           | 63.4            |
| stand          | 61.8      | 54.4    | 68.5           | 68.5           | 15.8            | 68.9           | 49.2           | 25.2            |
| swing baseball | 84.7      | 60.2    | 96.2           | 96.2           | 82.7            | 70.5           | 70.5           | 65.6            |
| throw          | 29.6      | 35.4    | 31.8           | 31.8           | 23.4            | 50.9           | 50.9           | 40.7            |
| walk           | 54.9      | 52.8    | 65.7           | 65.7           | 48.6            | 68.3           | 68.3           | 18.2            |
| wave           | 44.5      | 52.5    | 45.1           | 45.1           | 39.7            | 57.6           | 56.6           | 45.6            |
| avg            | 68.2      | 61.5    | 75.8           | 74.4           | 49.6            | 71.6           | 70.6           | 52.8            |

Table E.7: Frame-mAP and video-mAP per-class results J-HMDB splits 2 and 3.

# Bibliography

- Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Alexe, B., Deselaers, T., and Ferrari, V. (2010). What is an object? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Alexe, B., Deselaers, T., and Ferrari, V. (2012). Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Aljundi, R. and Tuytelaars, T. (2016). Lightweight unsupervised domain adaptation by convolutional filter reconstruction. In *Computer Vision, ECCV 2016 Workshops*, pages 508–515. Springer.
- All, K., Hasler, D., and Fleuret, F. (2011). Flowboostappearance learning from sparsely annotated video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1433–1440. IEEE.
- Aytar, Y. and Zisserman, A. (2011). Tabula rasa: Model transfer for object category detection. In *Proceedings of the International Conference on Computer Vision*, pages 2252–2259. IEEE.
- Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359.
- Benchmarks (2016). Benchmarks for popular cnn models. <https://github.com/jcjohnson/cnn-benchmarks>.
- Bergamo, A. and Torresani, L. (2010). Exploiting weakly-labeled web images to improve object classification: a domain adaptation approach. In *Advances in Neural Information Processing Systems*, pages 181–189.

- Bilen, H. and Vedaldi, A. (2016). Weakly supervised deep detection networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2846–2854.
- Blank, M., Gorelick, L., Shechtman, E., Irani, M., and Basri, R. (2005). Actions as space-time shapes. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1395–1402. IEEE.
- Bo, L., Ren, X., and Fox, D. (2011). Hierarchical matching pursuit for image classification: Architecture and fast algorithms. In *Advances in Neural Information Processing Systems*, volume 1, page 6.
- Bojanowski, P., Bach, F., Laptev, I., Ponce, J., Schmid, C., and Sivic, J. (2013). Finding actors and actions in movies. In *Proceedings of the International Conference on Computer Vision*.
- Bojanowski, P., Lajugie, R., Bach, F., Laptev, I., Ponce, J., Schmid, C., and Sivic, J. (2014). Weakly supervised action labeling in videos under ordering constraints. In *Proceedings of the European Conference on Computer Vision*, pages 628–643. Springer.
- Borgwardt, K. M., Gretton, A., Rasch, M. J., Kriegel, H.-P., Schlkopf, B., and Smola, A. J. (2006). Integrating structured biological data by kernel maximum mean discrepancy. In *Bioinformatics*.
- Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., and Krishnan, D. (2016a). Unsupervised pixel-level domain adaptation with generative adversarial networks. *arXiv preprint arXiv:1612.05424*.
- Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D., and Erhan, D. (2016b). Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351.
- Brand, M. (1999). Shadow puppetry. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1237–1244. IEEE.
- Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. (2004). High accuracy optical flow estimation based on a theory for warping. In *Proceedings of the European Conference on Computer Vision*.



- Brox, T. and Malik, J. (2010). Object segmentation by long term analysis of point trajectories. In *Proceedings of the European Conference on Computer Vision*.
- Brox, T. and Malik, J. (2011). Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Bucher, M., Herbin, S., and Jurie, F. (2016). Improving semantic embedding consistency by metric learning for zero-shot classification. In *Proceedings of the European Conference on Computer Vision*.
- Caba Heilbron, F., Carlos Niebles, J., and Ghanem, B. (2016). Fast temporal activity proposals for efficient detection of human actions in untrimmed videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1914–1923.
- Caesar, H., Uijlings, J., and Ferrari, V. (2015). Joint calibration for semantic segmentation. In *Proceedings of the British Machine Vision Conference*.
- Caesar, H., Uijlings, J., and Ferrari, V. (2016). Region-based semantic segmentation with end-to-end training. In *Proceedings of the European Conference on Computer Vision*, pages 381–397. Springer.
- Campbell, L. W., Becker, D. A., Azarbayejani, A., Bobick, A. F., and Pentland, A. (1996). Invariant features for 3-d gesture recognition. In *Proceedings of the International Conference on Automatic Face and Gesture Recognition*, pages 157–162. IEEE.
- Cao, L., Liu, Z., and Huang, T. S. (2010). Cross-dataset action detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. *arXiv preprint arXiv:1705.07750*.
- Chakravarty, P. and Tuytelaars, T. (2016). Cross-modal supervision for learning active speaker detection in video. In *Proceedings of the European Conference on Computer Vision*, pages 285–301. Springer.
- Chen, L., Li, W., and Xu, D. (2014a). Recognizing rgb images by learning from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1418–1425.

- Chen, W. and Corso, J. J. (2015). Action detection by implicit intentional motion clustering. In *Proceedings of the International Conference on Computer Vision*.
- Chen, W., Xiong, C., Xu, R., and Corso, J. (2014b). Actionness ranking with lattice conditional ordinal random fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Chen, W.-Y., Hsu, T.-M. H., Tsai, Y.-H. H., Wang, Y.-C. F., and Chen, M.-S. (2016). Transfer neural trees for heterogeneous domain adaptation. In *Proceedings of the European Conference on Computer Vision*, pages 399–414. Springer.
- Cherniavsky, N., Laptev, I., Sivic, J., and Zisserman, A. (2010). Semi-supervised learning of facial attributes in video. In *Proceedings of the European Conference on Computer Vision*, pages 43–56. Springer.
- Chopra, S., Balakrishnan, S., and Gopalan, R. (2013). Dlid: Deep learning for domain adaptation by interpolating between domains. In *ICML 2013, Workshop on Representation Learning, Atlanta, Georgia, USA*.
- Cinbis, R. G., Verbeek, J., and Schmid, C. (2014). Multi-fold MIL training for weakly supervised object localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Cinbis, R. G., Verbeek, J., and Schmid, C. (2016). Weakly supervised object localization with multi-fold multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Csurka, G., Dance, C., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, pages 1–2. Prague.
- Dalal, N. and Triggs, B. (2005). Histogram of Oriented Gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Daumé III, H. (2009). Frustratingly easy domain adaptation. *arXiv Preprint*.
- Del Pero, L., Ricco, S., Sukthankar, R., and Ferrari, V. (2016a). Behavior discovery and alignment of articulated object classes from unstructured video. *International Journal of Computer Vision*, pages 1–23.

- Del Pero, L., Ricco, S., Sukthankar, R., and Ferrari, V. (2016b). Discovering the physical parts of an articulated object class from multiple videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 714–723.
- Deselaers, T., Alexe, B., and Ferrari, V. (2012). Weakly supervised localization and learning with generic knowledge. *International Journal of Computer Vision*.
- Donahue, J., Hoffman, J., Rodner, E., Saenko, K., and Darrell, T. (2013). Semi-supervised domain adaptation with instance constraints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 668–675.
- Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, volume 32, pages 647–655.
- Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., and Brox, T. (2015). Flownet: Learning optical flow with convolutional networks. In *Proceedings of the International Conference on Computer Vision*, pages 2758–2766.
- Duan, L., Tsang, I. W., and Xu, D. (2012a). Domain transfer multiple kernel learning. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Duan, L., Tsang, I. W., Xu, D., and Maybank, S. J. (2009). Domain transfer svm for video concept detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1375–1381. IEEE.
- Duan, L., Xu, D., and Tsang, I. (2012b). Learning with augmented features for heterogeneous domain adaptation. In *International Conference on Machine Learning*.
- Duan, L., Xu, D., Tsang, I. W., and Luo, J. (2012c). Visual event recognition in videos by learning from web data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1667–1680.
- Duchenne, O., Laptev, I., Sivic, J., Bach, F., and Ponce, J. (2009). Automatic annotation of human actions in video. In *Proceedings of the International Conference on Computer Vision*, pages 1491–1498. IEEE.
- Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

- Escorcia, V., Heilbron, F. C., Niebles, J. C., and Ghanem, B. (2016). Daps: Deep action proposals for action understanding. In *Proceedings of the European Conference on Computer Vision*, pages 768–784. Springer.
- Escorcia, V., Niebles, J. C., and Ghanem, B. (2015). On the relationship between visual attributes and convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Everingham, M., Van Gool, L., Williams, C., J., W., and Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) Challenge 2010. *International Journal of Computer Vision*.
- Everingham, M., Van Gool, L., Williams, C., Winn, J., and Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 Results.
- Everingham, M., Van Gool, L., Williams, C., Winn, J., and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- Farhadi, A. and Tabrizi, M. K. (2008). Learning to recognize activities from the wrong view point. In *Proceedings of the European Conference on Computer Vision*, pages 154–166. Springer.
- Felzenszwalb, P., Girshick, R., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79.
- Fernando, B., Habrard, A., Sebban, M., and Tuytelaars, T. (2013). Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the International Conference on Computer Vision*, pages 2960–2967.
- Filipovych, R. and Ribeiro, E. (2008). Recognizing primitive interactions by exploring actor-object states. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE.
- Filipovych, R. and Ribeiro, E. (2011). Robust sequence alignment for actor–object interaction recognition: Discovering actor–object states. *Computer Vision and Image Understanding*, 115(2):177–193.

- Gaidon, A., Harchaoui, Z., and Schmid, C. (2013). Temporal localization of actions with actoms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2782–2795.
- Gaidon, A. and Vig, E. (2015). Online domain adaptation for multi-object tracking. In *Proceedings of the British Machine Vision Conference*.
- Gaidon, A., Zen, G., and Rodriguez-Serrano, J. A. (2014). Self-learning camera: Autonomous adaptation of object detectors to unlabeled video streams. *arXiv preprint arXiv:1406.4296*.
- Ganin, Y. and Lempitsky, V. (2015). Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*, pages 1180–1189.
- Gemert, J., Jain, M., Gati, E., Snoek, C. G., et al. (2015). APT: Action localization proposals from dense trajectories. In *Proceedings of the British Machine Vision Conference*.
- Ghifary, M., Kleijn, W. B., and Zhang, M. (2014). Domain adaptive neural networks for object recognition. In *Pacific Rim International Conference on Artificial Intelligence*, pages 898–904. Springer.
- Ghifary, M., Kleijn, W. B., Zhang, M., Balduzzi, D., and Li, W. (2016). Deep reconstruction-classification networks for unsupervised domain adaptation. In *Proceedings of the European Conference on Computer Vision*, pages 597–613. Springer.
- Gidaris, S. and Komodakis, N. (2015). Object detection via a multi-region and semantic segmentation-aware cnn model. In *Proceedings of the International Conference on Computer Vision*, pages 1134–1142.
- Girshick, R. (2015a). Fast R-CNN. In *Proceedings of the International Conference on Computer Vision*.
- Girshick, R. (2015b). Fast R-CNN. <https://github.com/rbgirshick/fast-rcnn/>.
- Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014a). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

- Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014b). Rich feature hierarchies for accurate object detection and semantic segmentation. <https://github.com/rbgirshick/rcnn/>.
- Girshick, R. B., Felzenszwalb, P. F., and McAllester, D. (2012a). Discriminatively trained deformable part models, release 5. <http://vision.stanford.edu/teaching/cs231b.spring1213/slides/dpmslidescross-girshick.pdf>.
- Girshick, R. B., Felzenszwalb, P. F., and McAllester, D. (2012b). Discriminatively trained deformable part models, release 5. <http://people.cs.uchicago.edu/rbg/latent-release5/>.
- Gkioxari, G. and Malik, J. (2015). Finding action tubes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Gong, B., Shi, Y., Sha, F., and Grauman, K. (2012). Geodesic flow kernel for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2066–2073. IEEE.
- Gonzalez-Garcia, A., Modolo, D., and Ferrari, V. (2016). Do semantic parts emerge in convolutional neural networks? *arXiv Preprint*.
- Gonzalez-Garcia, A., Modolo, D., and Ferrari, V. (2017). Objects as context for part detection. *arXiv Preprint*.
- Gonzalez-Garcia, A., Vezhnevets, A., and Ferrari, V. (2015). An active search strategy for efficient object class detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3022–3031.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.
- Gopalan, R., Li, R., and Chellappa, R. (2011). Domain adaptation for object recognition: An unsupervised approach. In *Proceedings of the International Conference on Computer Vision*, pages 999–1006. IEEE.
- Gopalan, R., Li, R., and Chellappa, R. (2014). Unsupervised adaptation across domain shifts by generating intermediate data representations. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

- Grundmann, M., Kwatra, V., Han, M., and Essa, I. (2010). Efficient hierarchical graph-based video segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Gu, C., Lim, J. J., Arbeláez, P., and Malik, J. (2009). Recognition using regions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1030–1037. IEEE.
- Gupta, A., Kembhavi, A., and Davis, L. S. (2009). Observing human-object interactions: Using spatial and functional compatibility for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Gupta, S., Arbeláez, P., Girshick, R., and Malik, J. (2015). Aligning 3d models to rgb-d images of cluttered scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4731–4740.
- Gupta, S., Girshick, R., Arbeláez, P., and Malik, J. (2014). Learning rich features from rgb-d images for object detection and segmentation. In *Proceedings of the European Conference on Computer Vision*, pages 345–360. Springer.
- Gupta, S., Hoffman, J., and Malik, J. (2016). Cross modal distillation for supervision transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2827–2836.
- Hartmann, G., Grundmann, M., Hoffman, J., Tsai, D., Kwatra, V., Madani, O., Vijayanarasimhan, S., Essa, I., Rehg, J., and Sukthankar, R. (2012). Weakly supervised learning of object segmentations from web-scale video. In *ECCV, Workshops and Demonstrations*, pages 198–208. Springer.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.
- Heilbron, F. C., Escorcia, V., Ghanem, B., and Nibbles, J. C. (2015). Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

- Henderson, P. and Ferrari, V. (2016). End-to-end training of object class detectors for mean average precision. *arXiv Preprint*.
- Hoai, M., Torresani, L., De la Torre, F., and Rother, C. (2014). Learning discriminative localization from weakly labeled data. *Pattern Recognition*, 47(3):1523–1534.
- Hoffman, J., Gupta, S., and Darrell, T. (2016). Learning with side information through modality hallucination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 826–834.
- Hoffman, J., Rodner, E., Donahue, J., Darrell, T., and Saenko, K. (2013). Efficient learning of domain-invariant image representations. In *International Conference on Learning Representations*.
- Hoffman, J., Rodner, E., Donahue, J., Kulis, B., and Saenko, K. (2014a). Asymmetric and category invariant feature transformations for domain adaptation. *International Journal of Computer Vision*.
- Hoffman, J., Tzeng, E., Donahue, J., Jia, Y., Saenko, K., and Darrell, T. (2014b). One-shot learning of supervised deep convolutional models. In *arXiv Preprint*, volume 2.
- Hogg, D. (1983). Model-based vision: a program to see a walking person. *Image and Vision Computing*, 1(1):5–20.
- Hou, R., Chen, C., and Shah, M. (2017). Tube convolutional neural network (t-cnn) for action detection in videos. *arXiv Preprint*.
- Hu, W., Tan, T., Wang, L., and Maybank, S. (2004). A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*.
- Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al. (2016a). Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv Preprint*.
- Huang, L., Yang, Y., Deng, Y., and Yu, Y. (2015). Densebox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*.
- Huang, Y.-H., Oramas, J., Tuytelaars, T., and Van Gool, L. (2016b). Do motion boundaries improve semantic segmentation?



- Jain, M., Van Gemert, J., Jégou, H., Bouthemy, P., and Snoek, C. G. (2014). Action localization with tubelets from motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 740–747.
- Jain, R., Militzer, D., and Nagel, H.-H. (1977). *Separating non-stationary from stationary scene components in a sequence of real world TV-images*. Institut für Informatik, Universität Hamburg.
- Jhuang, H., Gall, J., Zuffi, S., Schmid, C., and Black, M. J. (2013a). Towards understanding action recognition. In *Proceedings of the International Conference on Computer Vision*.
- Jhuang, H., Gall, J., Zuffi, S., Schmid, C., and Black, M. J. (2013b). Towards understanding action recognition. In *Proceedings of the International Conference on Computer Vision*, pages 3192–3199.
- Jia, Y. (2013). Caffe: An open source convolutional architecture for fast feature embedding. <http://caffe.berkeleyvision.org/>.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv Preprint*.
- Jiang, J. (2008). A literature survey on domain adaptation of statistical classifiers. <http://sifaka.cs.uiuc.edu/jiang4/domainadaptation/survey>.
- Kalogeiton, V., Schmid, C., and Ferrari, V. (2016). Analysing domain shift factors between videos and images for object detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Kalogeiton, V., Weinzaepfel, P., Ferrari, V., and Schmid, C. (2017a). Action tubelet detector for spatio-temporal action localization. In *Proceedings of the International Conference on Computer Vision*.
- Kalogeiton, V., Weinzaepfel, P., Ferrari, V., and Schmid, C. (2017b). Joint learning of object and action detectors. In *Proceedings of the International Conference on Computer Vision*.
- Kan, M., Shan, S., Zhang, H., Lao, S., and Chen, X. (2016). Multi-view discriminant analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):188–194.

- Kang, K., Li, H., Xiao, T., Ouyang, W., Yan, J., Liu, X., and Wang, X. (2017). Object detection in videos with tubelet proposal networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Kang, K., Li, H., Yan, J., Zeng, X., Yang, B., Xiao, T., Zhang, C., Wang, Z., Wang, R., Wang, X., et al. (2016a). T-cnn: Tubelets with convolutional neural networks for object detection from videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Kang, K., Ouyang, W., Li, H., and Wang, X. (2016b). Object detection from video tubelets with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Kantorov, V., Oquab, M., Cho, M., and Laptev, I. (2016). Contextlocnet: Context-aware deep network models for weakly supervised localization. In *Proceedings of the European Conference on Computer Vision*, pages 350–365. Springer.
- Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- kde (2003). Kernel density estimation toolbox for matlab (kde toolbox); silverman's rule of thumb. [www.ics.uci.edu/~ihler/code/kde.html](http://www.ics.uci.edu/~ihler/code/kde.html).
- Kim, G., Sigal, L., and Xing, E. P. (2014). Joint summarization of large sets of web images and videos for storyline reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Klaser, A., Marszałek, M., and Schmid, C. (2008). A Spatio-Temporal Descriptor Based on 3D-Gradients. In *Proceedings of the British Machine Vision Conference*, pages 275–1. British Machine Vision Association.
- Klaser, A., Marszałek, M., Schmid, C., and Zisserman, A. (2010). Human focused action localization in video. In *SGA 2010-International Workshop on Sign, Gesture, and Activity, ECCV 2010 Workshops*, volume 6553, pages 219–233. Springer.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

- Kroeger, T., Timofte, R., Dai, D., and Van Gool, L. (2016). Fast optical flow using dense inverse search. In *Proceedings of the European Conference on Computer Vision*.
- Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. (2011). Hmdb: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision*, pages 2556–2563. IEEE.
- Kulis, B., Saenko, K., and Darrell, T. (2011). What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1785–1792. IEEE.
- Kumar, M. P., Packer, B., and Koller, D. (2010). Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197.
- Kumar Singh, K., Xiao, F., and Jae Lee, Y. (2016). Track and transfer: Watching videos to simulate strong human supervision for weakly-supervised object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3548–3556.
- Kuo, W., Hariharan, B., and Malik, J. (2015). Deepbox: Learning objectness with convolutional networks. In *Proceedings of the International Conference on Computer Vision*, pages 2479–2487.
- Lampert, C. H., Nickisch, H., and Harmeling, S. (2009). Learning to detect unseen object classes by between-class attribute transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Lampert, C. H., Nickisch, H., and Harmeling, S. (2014). Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Lan, T., Wang, Y., and Mori, G. (2011). Discriminative figure-centric models for joint action localization and recognition. In *Proceedings of the International Conference on Computer Vision*, pages 2003–2010. IEEE.
- Lan, T., Zhu, Y., Roshan Zamir, A., and Savarese, S. (2015). Action recognition by hierarchical mid-level action elements. In *Proceedings of the International Conference on Computer Vision*.

- Laptev, I. (2005). On space-time interest points. *International Journal of Computer Vision*.
- Laptev, I. and Pérez, P. (2007). Retrieving actions in movies. In *Proceedings of the International Conference on Computer Vision*, pages 1–8.
- LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Lee, Y. J., Kim, J., and Grauman, K. (2011). Key-segments for video object segmentation. In *Proceedings of the International Conference on Computer Vision*.
- Leistner, C., Godec, M., Schulter, S., Saffari, A., and Bischof, H. (2011). Improving classifiers with weakly-related videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Li, D., Huang, J.-B., Li, Y., Wang, S., and Yang, M.-H. (2016a). Weakly supervised object localization with progressive domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3512–3520.
- Li, R. and Zickler, T. (2012). Discriminative virtual views for cross-view action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2855–2862. IEEE.
- Li, Y., He, K., Sun, J., et al. (2016b). R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, pages 379–387.
- Li, Y., Wang, N., Shi, J., Liu, J., and Hou, X. (2016c). Revisiting batch normalization for practical domain adaptation. *arXiv Preprint*.
- Li, Z., Gavves, E., Jain, M., and Snoek, C. G. (2016d). VideoLSTM convolves, attends and flows for action recognition. In *arXiv Preprint*.
- Liang, X., Liu, S., Wei, Y., Liu, L., Lin, L., and Yan, S. (2015). Towards computational baby learning: A weakly-supervised approach for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 999–1007.

- Lin, M., Chen, Q., and Yan, S. (2014a). Network in network. In *International Conference on Learning Representations*.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014b). Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision*.
- Liu, J., Kuipers, B., and Savarese, S. (2011). Recognizing human actions by attributes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Liu, M.-Y. and Tuzel, O. (2016). Coupled generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 469–477.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016a). SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016b). SSD: Single shot multibox detector. <https://github.com/weiliu89/caffe/tree/ssd>.
- Long, M., Cao, Y., Wang, J., and Jordan, M. (2015). Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, pages 97–105.
- Long, M., Wang, J., and Jordan, M. I. (2016a). Deep transfer learning with joint adaptation networks. *arXiv Preprint*.
- Long, M., Zhu, H., Wang, J., and Jordan, M. I. (2016b). Unsupervised domain adaptation with residual transfer networks. In *Advances in Neural Information Processing Systems*, pages 136–144.
- Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1150–1157. Ieee.
- Lu, C., Krishna, R., Bernstein, M., and Fei-Fei, L. (2016). Visual relationship detection with language priors. In *Proceedings of the European Conference on Computer Vision*.

- Ma, S., Zhang, J., Ikizler-Cinbis, N., and Sclaroff, S. (2013). Action recognition and localization by hierarchical space-time segments. In *Proceedings of the International Conference on Computer Vision*.
- Malisiewicz, T., Gupta, A., and Efros, A. (2011). Ensemble of exemplar-svms for object detection and beyond. In *Proceedings of the International Conference on Computer Vision*.
- Manen, S., Guillaumin, M., and Van Gool, L. (2013). Prime object proposals with randomized prim's algorithm. In *Proceedings of the International Conference on Computer Vision*, pages 2536–2543.
- Manen, S., Gygli, M., Dai, D., and Van Gool, L. (2017). Pathtrack: Fast trajectory annotation with path supervision. *arXiv preprint arXiv:1703.02437*.
- Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z., and Yuille, A. (2015). Deep captioning with multimodal recurrent neural networks (m-rnn). In *International Conference on Learning Representations*.
- Marian Puscas, M., Sangineto, E., Culibrk, D., and Sebe, N. (2015). Unsupervised tube extraction using transductive learning and dense trajectories. In *Proceedings of the International Conference on Computer Vision*, pages 1653–1661.
- Martin, W. N. and Aggarwal, J. (1977). Dynamic scene analysis: The study of moving images. Technical report, DTIC Document.
- Massa, F., Russell, B. C., and Aubry, M. (2016). Deep exemplar 2d-3d detection by adapting from real to rendered views. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6024–6033.
- Mathe, S. and Sminchisescu, C. (2012). Dynamic eye movement datasets and learnt saliency models for visual action recognition. *Proceedings of the European Conference on Computer Vision*, pages 842–856.
- Matikainen, P., Hebert, M., and Sukthankar, R. (2010). Representing pairwise spatial and temporal relations for action recognition. *Computer Vision—ECCV 2010*, pages 508–521.
- Messing, R., Pal, C., and Kautz, H. (2009). Activity recognition using the velocity histories of tracked keypoints. In *Proceedings of the International Conference on Computer Vision*, pages 104–111. IEEE.

- Mettes, P., van Gemert, J. C., and Snoek, C. G. (2016). Spot on: Action localization from pointly-supervised proposals. In *Proceedings of the European Conference on Computer Vision*, pages 437–453. Springer.
- Misra, I., Shrivastava, A., and Hebert, M. (2015). Watch and learn: Semi-supervised learning for object detectors from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3593–3602.
- Mittal, A., Raj, A., Namboodiri, V. P., and Tuytelaars, T. (2016). Unsupervised domain adaptation in the wild: Dealing with asymmetric label sets. *arXiv preprint arXiv:1603.08105*.
- Modolo, D. and Ferrari, V. (2016). Learning semantic part-based models from google images. *arXiv Preprint*.
- Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y. (2011). Multimodal deep learning. In *International Conference on Machine Learning*, pages 689–696.
- Nguyen, H. V., Ho, H. T., Patel, V. M., and Chellappa, R. (2015). Dash-n: Joint hierarchical domain adaptation and feature learning. *IEEE Transactions on Image Processing*, 24(12):5479–5491.
- Niebles, J. C., Chen, C.-W., and Fei-Fei, L. (2010). Modeling temporal structure of decomposable motion segments for activity classification. In *Proceedings of the European Conference on Computer Vision*, pages 392–405. Springer.
- Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the International Conference on Computer Vision*.
- Oh, S., Hoogs, A., Perera, A., Cuntoor, N., Chen, C.-C., Lee, J. T., Mukherjee, S., Aggarwal, J., Lee, H., and Davis, L. (2011a). A large-scale benchmark dataset for event recognition in surveillance video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3153–3160.
- Oh, S., Hoogs, A., Perera, A., Cuntoor, N., Chen, C.-C., Lee, J. T., Mukherjee, S., Aggarwal, J., Lee, H., Davis, L., et al. (2011b). A large-scale benchmark dataset for event recognition in surveillance video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

- Oliveira, G. L., Burgard, W., and Brox, T. (2016). Efficient deep models for monocular road segmentation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, pages 4885–4891. IEEE.
- Ommer, B., Mader, T., and Buhmann, J. M. (2009). Seeing the objects behind the dots: Recognition in videos from a moving camera. *International Journal of Computer Vision*, 83(1):57–71.
- Oneata, D., Revaud, J., Verbeek, J., and Schmid, C. (2014a). Spatio-temporal object detection proposals. In *Proceedings of the European Conference on Computer Vision*.
- Oneata, D., Verbeek, J., and Schmid, C. (2014b). Efficient action localization with approximately normalized fisher vectors. In *Proceedings of the International Conference on Computer Vision*, pages 2545–2552.
- Ouyang, W., Wang, X., Zeng, X., Qiu, S., Luo, P., Tian, Y., Li, H., Yang, S., Wang, Z., Loy, C.-C., et al. (2015). Deepid-net: Deformable deep convolutional neural networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2403–2412.
- Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Trans. on Knowledge and Data Engineering*.
- Pandey, M. and Lazebnik, S. (2011). Scene recognition and weakly supervised object localization with deformable part-based models. In *Proceedings of the International Conference on Computer Vision*.
- Papadopoulos, D. P., Clarke, A. D., Keller, F., and Ferrari, V. (2014). Training object class detectors from eye tracking data. In *Proceedings of the European Conference on Computer Vision*, pages 361–376. Springer.
- Papadopoulos, D. P., Uijlings, J. R., Keller, F., and Ferrari, V. (2016). We don't need no bounding-boxes: Training object class detectors using only human verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 854–863.
- Papadopoulos, D. P., Uijlings, J. R., Keller, F., and Ferrari, V. (2017). Training object class detectors with click supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.



- Papazoglou, A., Del Pero, L., and Ferrari, V. (2016a). Discovering object aspects from video. *Image and Vision Computing*, 52:206–217.
- Papazoglou, A., Del Pero, L., and Ferrari, V. (2016b). Video temporal alignment for object viewpoint. In *Proceedings of the Asian Conference on Computer Vision*, pages 273–288. Springer.
- Papazoglou, A. and Ferrari, V. (2013). Fast object segmentation in unconstrained video. In *Proceedings of the International Conference on Computer Vision*.
- Peng, X. and Schmid, C. (2016). Multi-region two-stream r-cnn for action detection. In *Proceedings of the European Conference on Computer Vision*.
- Perazzi, F., Pont-Tuset, J., McWilliams, B., Van Gool, L., Gross, M., and Sorkine-Hornung, A. (2016). A benchmark dataset and evaluation methodology for video object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 724–732.
- Perronnin, F., Sánchez, J., and Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. *Proceedings of the European Conference on Computer Vision*, pages 143–156.
- Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007). Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE.
- Phillips, P. J. and O’toole, A. J. (2014). Comparison of human and computer performance across face recognition experiments. *Image and Vision Computing*, 32(1):74–85.
- Pinheiro, P. O., Lin, T.-Y., Collobert, R., and Dollár, P. (2016). Learning to refine object segments. In *Proceedings of the European Conference on Computer Vision*.
- Pirsiavash, H., Ramanan, D., and Fowlkes, C. C. (2011). Globally-optimal greedy algorithms for tracking a variable number of objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1201–1208. IEEE.
- Prest, A., Ferrari, V., and Schmid, C. (2013). Explicit modeling of human-object interactions in realistic videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(4):835–848.

- Prest, A., Leistner, C., Civera, J., Schmid, C., and Ferrari, V. (2012a). Learning object class detectors from weakly annotated video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Prest, A., Schmid, C., and Ferrari, V. (2012b). Weakly supervised learning of interactions between humans and objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):601–614.
- Quadrianto, N. and Lampert, C. H. (2011). Learning multi-view neighborhood preserving projections. In *International Conference on Machine Learning*, pages 425–432.
- Raj, A., Namboodiri, V. P., and Tuytelaars, T. (2015). Subspace alignment based domain adaptation for rcnn detector. In *Proceedings of the British Machine Vision Conference*.
- Ramanan, D., Forsyth, D. A., and Barnard, K. (2006). Building models of animals from video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1319–1334.
- Raptis, M., Kokkinos, I., and Soatto, S. (2012). Discovering discriminative action parts from mid-level video representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788.
- Redmon, J. and Farhadi, A. (2016). Yolo9000: Better, faster, stronger. *arXiv Preprint*.
- Ren, S., He, K., Girshick, R., and Sun, J. (2015a). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*.
- Ren, S., He, K., Girshick, R., and Sun, J. (2015b). Faster R-CNN: Towards real-time object detection with region proposal networks. <https://github.com/rbgirshick/py-faster-rcnn/>.
- Ren, S., He, K., Girshick, R., Zhang, X., and Sun, J. (2016). Object detection networks on convolutional feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

- Rodriguez, M. D., Ahmed, J., and Shah, M. (2008). Action MACH: a spatio-temporal maximum average correlation height filter for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Rogez, G., Weinzaepfel, P., and Schmid, C. (2017). LCR-Net: Localization-classification-regression for human pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Rohr, K. (1994). Towards model-based recognition of human movements in image sequences. *CVGIP: Image Understanding*, 59(1):94–115.
- Rozantsev, A., Salzmann, M., and Fua, P. (2016). Beyond sharing weights for deep domain adaptation. *arXiv Preprint*.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2013a). Imagenet large scale visual recognition challenge (ilsvrc). <http://www.image-net.org/challenges/LSVRC/2012>.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2013b). Imagenet large scale visual recognition challenge (ilsvrc). <http://www.image-net.org/challenges/LSVRC/2013>.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2014). Imagenet large scale visual recognition challenge (ilsvrc). <http://www.image-net.org/challenges/LSVRC/2014>.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015a). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015b). Imagenet large scale visual recognition challenge (ilsvrc). <http://www.image-net.org/challenges/LSVRC/2015>.

- Sadeghi, M. A. and Farhadi, A. (2011). Recognition using visual phrases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Saenko, K., Kulis, B., Fritz, M., and Darrell, T. (2010). Adapting visual category models to new domains. *Proceedings of the European Conference on Computer Vision*, pages 213–226.
- Saha, S., Singh, G., and Cuzzolin, F. (2017). Amtnet: Action-micro-tube regression by end-to-end trainable deep architecture. *arXiv Preprint*.
- Saha, S., Singh, G., Sapienza, M., Torr, P. H., and Cuzzolin, F. (2016). Deep learning for detecting multiple space-time action tubes in videos. In *Proceedings of the British Machine Vision Conference*.
- Sánchez, J., Perronnin, F., Mensink, T., and Verbeek, J. (2013). Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245.
- Satkin, S. and Hebert, M. (2010). Modeling the temporal extent of actions. *Proceedings of the European Conference on Computer Vision*, pages 536–548.
- Schuldt, C., Laptev, I., and Caputo, B. (2004). Recognizing human actions: A local svm approach. In *Proceedings of the International Conference on Pattern Recognition*, volume 3, pages 32–36. IEEE.
- Schüldt, C., Laptev, I., and Caputo, B. (2004). Recognizing human actions: a local svm approach. In *Proceedings of the International Conference on Pattern Recognition*.
- Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations*.
- Sharma, A., Kumar, A., Daume, H., and Jacobs, D. W. (2012a). Generalized multiview analysis: A discriminative latent space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2160–2167. IEEE.
- Sharma, P., Huang, C., and Nevatia, R. (2012b). Unsupervised incremental learning for improved object detection in a video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3298–3305. IEEE.

- Sharma, P. and Nevatia, R. (2013). Efficient detector adaptation for object detection in a video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Shi, M., Caesar, H., and Ferrari, V. (2017). Weakly supervised object localization using things and stuff transfer. *arXiv Preprint*.
- Shi, M. and Ferrari, V. (2016). Weakly supervised object localization using size estimates. In *Proceedings of the European Conference on Computer Vision*, pages 105–121. Springer.
- Shu, X., Qi, G.-J., Tang, J., and Wang, J. (2015). Weakly-shared deep transfer networks for heterogeneous-domain knowledge propagation. In *Proceedings of the ACM International Conference on Multimedia*, pages 35–44. ACM.
- Simonyan, K. and Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*.
- Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.
- Singh, G., Saha, S., Sapienza, M., Torr, P., and Cuzzolin, F. (2017). Online real time multiple spatiotemporal action localisation and prediction on a single platform. In *arXiv Preprint*.
- Song, H., Girshick, R., Jegelka, S., Mairal, J., Harchaoui, Z., and Darrell, T. (2014). On learning to localize objects with minimal supervision. In *International Conference on Machine Learning*.
- Song, S. and Xiao, J. (2014). Sliding shapes for 3d object detection in depth images. In *Proceedings of the European Conference on Computer Vision*, pages 634–651. Springer.
- Soomro, K., Zamir, A. R., and Shah, M. (2012). UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. In *CRCV-TR-12-01*.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

- Su, H., Deng, J., and Fei-Fei, L. (2012). Crowdsourcing annotations for visual object detection. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, volume 1.
- Sun, B. and Saenko, K. (2016). Deep coral: Correlation alignment for deep domain adaptation. In *Computer Vision—ECCV 2016 Workshops*, pages 443–450. Springer.
- Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv Preprint*.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. In *arXiv Preprint*.
- Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708.
- Tang, K., Ramanathan, V., Fei-Fei, L., and Koller, D. (2012). Shifting weights: Adapting object detectors from image to video. In *Advances in Neural Information Processing Systems*.
- Tang, K., Sukthankar, R., Yagnik, J., and Fei-Fei, L. (2013). Discriminative segment annotation in weakly labeled video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Tian, Y., Sukthankar, R., and Shah, M. (2013). Spatiotemporal deformable part models for action detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2642–2649.
- Tokmakov, P., Alahari, K., and Schmid, C. (2016). Weakly supervised semantic segmentation using motion cues. In *Proceedings of the European Conference on Computer Vision*, pages 388–404. Springer.
- Tokmakov, P., Alahari, K., and Schmid, C. (2017a). Learning motion patterns in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Tokmakov, P., Alahari, K., and Schmid, C. (2017b). Learning video object segmentation with visual memory. *arXiv Preprint*.

- Torralba, A. and Efros, A. A. (2011). Unbiased look at dataset bias. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Tran, D. and Yuan, J. (2011). Optimal spatio-temporal path discovery for video event detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3321–3328. IEEE.
- Tran, D. and Yuan, J. (2012). Max-margin structured output regression for spatio-temporal action localization. In *Advances in Neural Information Processing Systems*, pages 350–358.
- Tzeng, E., Hoffman, J., Darrell, T., and Saenko, K. (2015). Simultaneous deep transfer across domains and tasks. In *Proceedings of the International Conference on Computer Vision*, pages 4068–4076.
- Tzeng, E., Hoffman, J., Saenko, K., and Darrell, T. (2017). Adversarial discriminative domain adaptation. *arXiv preprint arXiv:1702.05464*.
- Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., and Darrell, T. (2014). Deep domain confusion: Maximizing for domain invariance. *arXiv Preprint*.
- Uijlings, J. R., van de Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective search for object recognition. *International Journal of Computer Vision*.
- Urmson, C. (2015). How a driverless car sees the road. [https://www.ted.com/talks/chris\\_urmson\\_how\\_a\\_driverless\\_car\\_sees\\_the\\_road](https://www.ted.com/talks/chris_urmson_how_a_driverless_car_sees_the_road).
- Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. In *jmlr*.
- Van Gemert, J. C., Veenman, C. J., Smeulders, A. W., and Geusebroek, J.-M. (2010). Visual word ambiguity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1271–1283.
- Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., and Saenko, K. (2015). Sequence to sequence-video to text. In *Proceedings of the International Conference on Computer Vision*.
- Vezhnevets, A. and Ferrari, V. (2015). Object localization in imagenet by looking out of the window. In *Proceedings of the British Machine Vision Conference*.

- Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, pages 1096–1103. ACM.
- Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269.
- Volpi, M. and Ferrari, V. (2015). Semantic segmentation of urban scenes by learning local class interactions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–9.
- Wang, A., Lu, J., Cai, J., Cham, T.-J., and Wang, G. (2015a). Large-margin multi-modal deep learning for rgb-d object recognition. *IEEE Transactions on Multimedia*, 17(11):1887–1898.
- Wang, H., Kläser, A., Schmid, C., and Liu, C.-L. (2013). Dense trajectories and motion boundary descriptors for action recognition. *International Journal of Computer Vision*, 103(1):60–79.
- Wang, H., Oneata, D., Verbeek, J., and Schmid, C. (2015b). A robust and efficient video representation for action recognition. *International Journal of Computer Vision*.
- Wang, L., Qiao, Y., and Tang, X. (2014). Video action detection with relational dynamic-poselets. In *Proceedings of the European Conference on Computer Vision*, pages 565–580. Springer.
- Wang, L., Qiao, Y., Tang, X., and Van Gool, L. (2016a). Actionness estimation using hybrid fully convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.



- Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., and Van Gool, L. (2016b). Temporal segment networks: towards good practices for deep action recognition. In *Proceedings of the European Conference on Computer Vision*.
- Wang, X. and Gupta, A. (2015). Unsupervised learning of visual representations using videos. In *Proceedings of the International Conference on Computer Vision*, pages 2794–2802.
- Wang, X., Hua, G., and Han, T. X. (2012). Detection by detections: Non-parametric detector adaptation for a video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 350–357. IEEE.
- Weinzaepfel, P., Harchaoui, Z., and Schmid, C. (2015). Learning to track for spatio-temporal action localization. In *Proceedings of the International Conference on Computer Vision*.
- Weinzaepfel, P., Martin, X., and Schmid, C. (2017). Human action localization with sparse spatial supervision. *arXiv preprint arXiv:1605.05197*.
- Xu, C. and Corso, J. J. (2012). Evaluation of super-voxel methods for early video processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1202–1209. IEEE.
- Xu, C. and Corso, J. J. (2016). Actor-action semantic segmentation with grouping-process models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Xu, C., Hsieh, S.-H., Xiong, C., and Corso, J. J. (2015). Can humans fly? Action understanding with multiple classes of actors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Xu, J., Ramos, S., Vázquez, D., and Lopez, A. M. (2014). Domain adaptation of deformable part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(12):2367–2380.
- Yamato, J., Ohya, J., and Ishii, K. (1992). Recognizing human action in time-sequential images using hidden markov model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 379–385. IEEE.

- Yang, J., Yan, R., and Hauptmann, A. G. (2007a). Adapting svm classifiers to data with shifted distributions. In *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on*, pages 69–76. IEEE.
- Yang, J., Yan, R., and Hauptmann, A. G. (2007b). Cross-domain video concept detection using adaptive svms. In *Proceedings of the ACM International Conference on Multimedia*, pages 188–197. ACM.
- Yao, B., Jiang, X., Khosla, A., Lin, A. L., Guibas, L., and Fei-Fei, L. (2011). Human action recognition by learning bases of action attributes and parts. In *Proceedings of the International Conference on Computer Vision*.
- Yao, L., Torabi, A., Cho, K., Ballas, N., Pal, C., Larochelle, H., and Courville, A. (2015). Describing videos by exploiting temporal structure. In *Proceedings of the International Conference on Computer Vision*.
- Yoo, D., Park, S., Lee, J.-Y., Paek, A. S., and So Kweon, I. (2015). Attentionnet: Aggregating weak directions for accurate object detection. In *Proceedings of the International Conference on Computer Vision*, pages 2659–2667.
- Yu, G. and Yuan, J. (2015). Fast action proposals for human action detection and search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Yuan, J., Liu, Z., and Wu, Y. (2009). Discriminative subvolume search for efficient action detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2442–2449. IEEE.
- Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pages 818–833. Springer.
- Zhang, K., Chao, W.-L., Sha, F., and Grauman, K. (2016). Summary transfer: Exemplar-based subset selection for video summarization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1059–1067.
- Zhu, X., Wang, Y., Dai, J., Yuan, L., and Wei, Y. (2017a). Flow-guided feature aggregation for video object detection. *arXiv Preprint*.

- Zhu, X., Xiong, Y., Dai, J., Yuan, L., and Wei, Y. (2017b). Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Zitnick, C. L. and Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *Proceedings of the European Conference on Computer Vision*, pages 391–405. Springer.
- Zolfaghari, M., Oliveira, G. L., Sedaghat, N., and Brox, T. (2017). Chained multi-stream networks exploiting pose, motion, and appearance for action classification and detection. *arXiv Preprint*.