This thesis focuses on the problem of object tracking. Given a video, the general objective of tracking is to track the location over time of one or more targets in the image sequence. This is a very challenging task as algorithms need to deal with problems such as appearance variations, non-rigid deformations, cluttered background, occlusions etc. While most existing methods use bounding boxes to represent the target, we use segmentations instead, which provide better access to target pixels and can better handle occlusions. Our first contribution, is a new tracking algorithm that given an over-segmentation of a video tracks multiple targets through interactions and occlusions. We develop a provably convergent learning algorithm for this approach, which leverages training data to improve performance. Our second contribution targets the case when an over-segmentation is not available due to poor video quality or low resolution. For this case, we develop a new algorithm that tracks coherent regions and estimates the number of target objects in each region. This count representation of a video can be used to help inform more traditional tracking techniques. Finally, we develop the first tracking-by-segmentation approach based on deep learning. We propose a novel deep network architecture and training algorithms for learning to segment and track a target object throughout a video. All of our algorithms are rigorously evaluated on challenging benchmark video collections, which demonstrate improvements over the state-of-the-art.
Object Tracking-by-Segmentation in Videos

by

Sheng Chen

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented March 7, 2017
Commencement June 2017

APPROVED:

______________________________
Major Professor, representing Computer Science

______________________________
Director of the School of Electrical Engineering and Computer Science

______________________________
Dean of the Graduate School

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

______________________________
Sheng Chen, Author
ACKNOWLEDGEMENTS

I would like to first express my appreciation to my advisor Prof Fern who has been very helpful and supportive throughout my PhD study. I would like to thank you for giving me the opportunity to start this PhD journey in the first place. I learned a lot from you, not only some specific algorithms but also the way of thinking to approach any research problem.

I would also like to thank Prof Todorovic. I always treat you as my co-advisor and I’m very thankful to all your professional advices in computer vision and also in general.

I would like to thank the rest of my committee members: Prof. Li, Prof. Niemeyer and Prof. Tadepalli for your precious time.

I would like to thank my friends and our vision group members. Discussions with you always help me a lot in both research and life.

Last but not the least, I would like to thank my parents who have always been very understanding and supportive.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Prior Work</td>
</tr>
<tr>
<td>2.1</td>
<td>ONLINE TRACKING</td>
</tr>
<tr>
<td>2.2</td>
<td>OFFLINE TRACKING-BY-DETECTION</td>
</tr>
<tr>
<td>2.3</td>
<td>TRACKING-BY-SEGMENTATION</td>
</tr>
<tr>
<td>2.4</td>
<td>DEEP NETWORKS FOR TRACKING</td>
</tr>
<tr>
<td>3</td>
<td>Over-Segmentation to Tracking</td>
</tr>
<tr>
<td>3.1</td>
<td>INTRODUCTION</td>
</tr>
<tr>
<td>3.1.1</td>
<td>VIDEO FEATURES</td>
</tr>
<tr>
<td>3.1.2</td>
<td>DATA ASSOCIATION</td>
</tr>
<tr>
<td>3.1.3</td>
<td>OUR APPROACH: CSL</td>
</tr>
<tr>
<td>3.2</td>
<td>CONSTRAINED SEQUENTIAL LABELING</td>
</tr>
<tr>
<td>3.2.1</td>
<td>MAIN ALGORITHM</td>
</tr>
<tr>
<td>3.2.2</td>
<td>SEQUENTIAL LABELING WITH A COST FUNCTION</td>
</tr>
<tr>
<td>3.2.3</td>
<td>INEQUALITY CONSTRAINTS AND PROPAGATION</td>
</tr>
<tr>
<td>3.2.4</td>
<td>HANDLING UNKNOWN NUMBERS OF OBJECTS</td>
</tr>
<tr>
<td>3.3</td>
<td>COST FUNCTION REPRESENTATION AND LEARNING</td>
</tr>
<tr>
<td>3.3.1</td>
<td>COST FUNCTION DESCRIPTORS</td>
</tr>
<tr>
<td>3.3.2</td>
<td>COST FUNCTION LEARNING</td>
</tr>
<tr>
<td>3.3.3</td>
<td>CONVERGENCE OF LEARNING</td>
</tr>
<tr>
<td>3.4</td>
<td>CONSTRAINT LEARNING</td>
</tr>
<tr>
<td>3.5</td>
<td>EXPERIMENTAL RESULTS</td>
</tr>
<tr>
<td>3.6</td>
<td>CONCLUSION</td>
</tr>
<tr>
<td>4</td>
<td>Under-Segmentation to Tracking</td>
</tr>
<tr>
<td>4.1</td>
<td>INTRODUCTION</td>
</tr>
<tr>
<td>4.2</td>
<td>PRIOR WORK</td>
</tr>
<tr>
<td>4.3</td>
<td>APPROACH</td>
</tr>
<tr>
<td>4.3.1</td>
<td>OVERVIEW</td>
</tr>
<tr>
<td>4.3.2</td>
<td>FOREGROUND DETECTION AND GRAPH BUILDING</td>
</tr>
<tr>
<td>4.3.3</td>
<td>INTEGER PROGRAMMING FORMULATION</td>
</tr>
<tr>
<td>4.3.4</td>
<td>ERROR-DRIVEN GRAPH REVISION</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4 EXPERIMENTAL RESULTS</td>
<td>47</td>
</tr>
<tr>
<td>4.4.1 EVALUATION METRICS</td>
<td>48</td>
</tr>
<tr>
<td>4.4.2 RESULTS</td>
<td>49</td>
</tr>
<tr>
<td>4.5 CONCLUSION</td>
<td>51</td>
</tr>
<tr>
<td>5 Deep Tracking-by-Segmentation</td>
<td>57</td>
</tr>
<tr>
<td>5.1 INTRODUCTION</td>
<td>57</td>
</tr>
<tr>
<td>5.2 RELATED WORK</td>
<td>58</td>
</tr>
<tr>
<td>5.3 PROBLEM STATEMENT AND OVERVIEW</td>
<td>58</td>
</tr>
<tr>
<td>5.4 DEEP LEARNING FOR TRACKING-BY-SEGMENTATION</td>
<td>59</td>
</tr>
<tr>
<td>5.4.1 DEEP NETWORK ARCHITECTURE</td>
<td>59</td>
</tr>
<tr>
<td>5.4.2 TRAINING</td>
<td>60</td>
</tr>
<tr>
<td>5.5 EXPERIMENTAL RESULTS</td>
<td>63</td>
</tr>
<tr>
<td>5.6 CONCLUSION</td>
<td>65</td>
</tr>
<tr>
<td>6 Conclusion and Future Directions</td>
<td>73</td>
</tr>
<tr>
<td>Bibliography</td>
<td>74</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>An example result from our first approach on a single frame. Each color indicates one target of interest.</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Count localization problem. We would like to output a set of detections (a mix of bounding boxes and segments) and also predict the number of people inside each detection. Here all detections are shown using bounding boxes for better illustration.</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Results of our deep tracking-by-segmentation network. The goal here is to get an accurate object boundary.</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>Bounding boxes of the DPM detector [34] (left), and supervoxels of [103] (right). The DPM often misses players in crowds (top) and in unexpected poses (bottom). Supervoxels are more suitable for our challenging videos, because their boundaries are usually aligned with part boundaries facilitating bottom up reasoning.</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of sequential labeling of supervoxels. Blue rectangles represent already labeled supervoxels, the neighboring shaded rectangles are active supervoxels that will be considered for labeling, the remaining rectangles will not be considered until they become active.</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Illustration of determining an inequality constraint, that is, whether two supervoxels can have the same label. In frame $t$, $S_i$ and $S_j$ belong to a single foreground connected component, so the frame does not yield an inequality constraint. In frame $t'$, the SVM classifier decides the two connected components cannot belong to a single individual, generating an inequality constraint between $S_i$ and $S_j$.</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Left: supervoxels that falls in the foreground mask. Different colors indicate different supervoxels. Right: each point trajectory is shown by a red circle.</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>ROC curve for the constraint classifier when either using the spatial layout of supervoxels, or raw optical flow as input features. In particular, for precision of 97.3%, the former recall is 44.57%, and the latter recall is 8.04%. For generating responses of our constraint classifier, we operate in the region of high precision rates.</td>
<td>34</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES (Continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6</td>
<td>One failure example. Here the orange player occludes the yellow player, but our approach labels most of the orange player yellow. One reason is that our background subtraction removes the supervoxel corresponding to the player’s leg from our list of foreground supervoxels. At the same time, the two players are both moving left, so none of our features used for computing affinities between supervoxels provides useful information.</td>
<td>35</td>
</tr>
<tr>
<td>3.7</td>
<td>Volleyball dataset: first row(CSL-VOX), second row(Ncut), third row(Superpixel), fourth row(Network Flow*).</td>
<td>36</td>
</tr>
<tr>
<td>3.8</td>
<td>Our tracking results in the Figment dataset.</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Our count localization results for an image sequence from American football. Challenges include severe occlusion, clutter, and similar appearance of players.</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>System Overview. First extract noisy foreground by running an object detector and foreground segmentation. A flow graph is then built and transformed into an integer program. In the following iterations, the approach detects places where integrity/domain constraints are violated by the current solution and applies one or more graph-revision operators to obtain a new graph and updated solution.</td>
<td>42</td>
</tr>
<tr>
<td>4.3</td>
<td>Example results of object detector and foreground segmentation. Left: foreground segmentation, Right: DPM detector. The segmentation misses several isolated small players while the DPM misses players in the crowd.</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Illustration of three operators. Top: Add a node. Miss a detection in one frame while there are corresponding detections in neighboring frames. Middle: Add a tracker. A target continue missing for several frames. Bottom: Add an edge. This should be a merge, but we did not connect initially because of inappropriate threshold.</td>
<td>53</td>
</tr>
<tr>
<td>4.5</td>
<td>Histogram of counts for football (left) and UCSD (right) dataset.</td>
<td>54</td>
</tr>
<tr>
<td>4.6</td>
<td>Heat map of counts for the pedestrian dataset. Left: left direction, Right: right direction. See Fig.4.8 for image of scene.</td>
<td>54</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.7</td>
<td>An example image sequence from American football dataset. First row: initial</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>input of foreground detections, second row: ([86]), third row: (EGR₁),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fourth row (EGR). [86] and EGR₁ miss a few players because of the initial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>input. EGR is able to recover these miss detections by iteratively revise</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the graph.</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>Pedestrian dataset. Green and Red boundaries outline the foreground blobs</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>moving left and right. Green bounding boxes show smaller detection from</td>
<td></td>
</tr>
<tr>
<td></td>
<td>greedy assignment.</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Our CNN architecture. Note that we remove the pooling layers which is</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>widely used in other CNNs. We empirically found that this change improves</td>
<td></td>
</tr>
<tr>
<td></td>
<td>results for our problem. Such non-pooling networks have also been used for</td>
<td></td>
</tr>
<tr>
<td></td>
<td>saliency detection such as [63].</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Our unrolled RNN. $m^i_t$ denotes the mask from $i$th iteration where the</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>initial mask $m^0_t$ comes from our CNN. $\Delta^i_t$ denotes the modification</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$e_t$ denotes the edge image we extract in advance to help improve the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>boundary.</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Results on SegTrackV2 dataset</td>
<td>71</td>
</tr>
<tr>
<td>5.4</td>
<td>Results on Youtube dataset. For each sequence, the second row shows our</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>results and the third row shows results of [99].</td>
<td></td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>VolleyBall dataset. Network Flow* is an oracle method that uses ground truth detections.</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>VolleyBall dataset. Results of the proposed learning algorithm using both un-normalized and normalized features.</td>
<td>32</td>
</tr>
<tr>
<td>3.3</td>
<td>VolleyBall dataset. Results of different variants of CSL.</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>Results for the Figment dataset. PRCE: percentage of wrongly labeled pixels per player mask; Recall: percentage of recalled pixels per player mask; Tracking time: the number of frames where recall is above 20%.</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>Results for PETS2009 S2L1</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>Results of tracking accuracy and running time of our approach with different number of supervoxels as input for videos with 300 frames.</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Results for American Football dataset. Maximum number of targets is 23 per frame. To provide a better understanding of our CLA metric. Frame is an oracle frame-level approach.</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>Results for Pedestrian dataset. * We calculated CLA for [10, 26, 66] assuming they output 2 detections, one for each direction, and each direction has the correct counts.</td>
<td>51</td>
</tr>
<tr>
<td>5.1</td>
<td>Results for the SegTrackv2 dataset. We describe each method with two attributes. Generic means that the method is not trained for specific class and can be applied to generic object. Deep indicates whether the method ever uses any deep networks.</td>
<td>69</td>
</tr>
<tr>
<td>5.2</td>
<td>Results for the Youtube dataset.</td>
<td>70</td>
</tr>
<tr>
<td>5.3</td>
<td>Results compared with baselines</td>
<td>70</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>1 Constrained Sequential Labeling Algorithm with learning option. When</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>the flag learn is set, the algorithm performs one run of learning the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight vector $w$. . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Inequality constraint generation. . . . . . . . . . . . . . . . . . .</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>3 Our CNN training algorithm for one video . . . . . . . . . . . . . . .</td>
<td>63</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

This thesis focuses on the problem of object tracking. Tracking is one of the most active research areas in computer vision. Given a video, the general objective of tracking is to track the location over time of one or more targets in the image sequence. Tracking is a very important problem as it is a crucial step for high level vision tasks like activity recognition and also has its own applications like video surveillance. Tracking is very challenging as algorithms need to deal with problems such as appearance variations, non-rigid deformations, cluttered background and occlusion.

A large amount of work has been done in both single object tracking [69, 70, 112, 65, 108] and multiple objects tracking [104, 90, 46, 6, 102, 60, 14]. Most of these works output bounding boxes as target representations where each bounding box is at most a 6-dimensional vector with x, y coordinates, width, height and angle. While bounding boxes are sufficient in some cases, they can also be problematic. For example, if we need to operate on the pixels of the target(s), we then need a precise mask of the target. In addition, when there is occlusion, bounding boxes also cause problems as it is hard to distinguish the visible part of the target from the occluder within the bounding box.

In order to address the shortcomings of bounding box representations, in this thesis we propose to track objects by tracking their outlines, which is referred to as tracking-by-segmentation. Instead of using bounding boxes to represent the target, tracking-by-segmentation represents each target of interest as a segment. The problem can be precisely defined as given a video, label each pixel as a particular target or background.

Traditionally directly labeling each pixel is usually time consuming, so we need to ground our tracking algorithms onto something else, i.e certain pre-computed segmentations. Segmentation is also a significant area of computer vision research. The goal of segmentation is generally to partition the image/video into several regions where pixels inside each region share certain properties depending on the specific task. For example, pixels inside a region can have similar motion/appearance, or have the same semantic label (certain class, background, etc.). While there are various ways to further categorize segmentation problems, one possible answer is over-segmentation vs under-segmentation. In over-segmentation, one object is usually further
In this thesis, we propose three tracking-by-segmentation algorithms.\textsuperscript{1} The first work \cite{22} tackles the problem of multi-object tracking-by-segmentation. In this work, given a video and a set of targets of interests in the first frame, our goal is to label each pixel in the following frames with target ids. An example result is shown in Fig. 1.1. We start from an over-segmentation of the video and developed a sequential algorithm called constrained sequential labeling (CSL) that labels over-segmented features using a learned cost function and a set of learned constraints that come from domain knowledge. We show that this method can handle occlusions and target interactions well when we can get a relatively good over-segmentation of the video. However, when a good over-segmentation is hard to get, which is often the case with videos of low resolution and the targets being relatively small and crowded, this method is very likely to fail.

Thus, in the second work \cite{23}, instead of assuming there is a good over-segmentation of the video, we work with under-segmentations. In this work, we define a new problem called count localization (see Fig. 1.2) where for a given video the goal is to output a set of detections and

\textsuperscript{1}Other works that are done during the PhD can be found at \cite{21, 68}.
Figure 1.2: Count localization problem. We would like to output a set of detections (a mix of bounding boxes and segments) and also predict the number of people inside each detection. Here all detections are shown using bounding boxes for better illustration.

for each detection, output a count indicating the number of people inside the detection. We start from a set of noisy detections of people and a foreground segmentation and developed a framework called error-driven graph revision (EGR). We formulate our count localization problem using a flow graph and derive an integer program (IP) to solve the problem. Since the initial input detections and foreground are all noisy, the algorithm checks whether there are domain constraints violated in the IP solution and proposes ways to fix the graph accordingly which gives us a new IP to solve. This process is iterated until no domain constraints are violated. While this algorithm does not solve the tracking problem directly, the resulting under-segmented features with counts can be used for tracking as suggested in prior work[73, 42].

Finally, with the recent advance in neural networks, we propose a deep network architecture that provides an end-to-end solution to tracking-by-segmentation of a single target (see Fig. 1.3). Our network is composed of a convolutional neural network (CNN) and a recurrent neural network (RNN) where the CNN takes the current frame, previous frame and previous target mask as input and predicts a rough target mask in the current frame. The RNN then takes the CNN output and an additional edge image as input and iteratively refines the target mask to improve the boundary. The networks are applied sequentially to each frame which raises a challenge for

\[\text{submitted to CVPR 2017.}\]
learning as the learning algorithm needs to account for the dependence between the predictions. To solve this problem, we also propose a learning framework that iteratively collects training data for our network.

In the following chapters, we first review some common prior works on object tracking in Chapter 2. Additional related work will also be covered each each chapter. In Chapter 3, we describe our constrained sequential labeling (CSL) algorithm that tracks target segments based on an over-segmentation of the video. In Chapter 4, we presents our error-driven graph revision (EGR) framework that given an under-segmentation of the video (e.g. a foreground mask), computes a count of the targets in each segment. In Chapter 5, a deep network architecture is proposed to solve tracking-by-segmentation problem which to the best of our knowledge is the first deep network that solves the problem. Finally, in Chapter 6, we draw general conclusions and point to future work.
Chapter 2: Prior Work

Tracking has attracted significant attention due to its wide variety of applications. In this thesis, we present four categories of prior work. We first review work on online tracking. In this setting, the algorithms only have access to frames before the current frame. Most online tracking work focuses on how to better represent the target(s) and how to online update the target model(s). In the second part, we review approaches that work with the entire video e.g. offline tracking. In this case, it is typical to first run an object detector on all frames and formulate tracking as a data association problem. While most approaches in both settings use bounding boxes, there has also been prior approaches that try to track the outline of the target(s). We review that work in the third part. Finally, we introduce some of the most recent work, which explores deep learning for tracking.

2.1 ONLINE TRACKING

Many tracking approaches operate on a frame-to-frame basis where the decisions in the current frame only depend on the previous frames. Examples of this approach include early work using Kalman filters [82, 47] and more recent work utilizing particle filters [17, 95]. A challenge for these approaches is that their state space usually grows exponentially with the number of objects. A naive solution is to use an independent (particle) filters for each object. This approach, however, often leads to filter “hijacking”, where multiple filters track a single prominent object.

Several approaches have been introduced to deal with the hijacking problem. [75] uses an object detector during and after interactions to help avoid hijacking. Alternatively, [55] tracks objects in their joint state space and uses a Markov random field to enforce separation between nearby objects. Rather, [43] uses “pseudo-independent” log-linear filters, where each object has its own particle filter to estimate the current state, but the filters have access to the previous estimated states of other filters. While these approaches have successfully dealt with hijacking in some situation, none of them explicitly attempt to model occlusion. In contrast, a recent approach [105] introduces occlusion variables for each object and jointly estimates the occlusion variables with the states of the objects. This approach, however, has difficulty dealing with
complicated occlusions that involve more than two objects. Our approach attempts to improve on these approaches by working with supervoxels, which provide information spanning more than just neighboring frames. This combined with our constraint reasoning helps to more effectively resolve occlusions.

2.2 OFFLINE TRACKING-BY-DETECTION

This line of research operates in an offline fashion where the entire video is given beforehand. To restrict the solution space, an object detector is usually run to detect potential locations of objects in all frames [46, 39, 100, 6, 102, 58, 44, 81, 18, 45], or to limit the object locations to a regular grid [13, 8, 101]. Tracking is then formulated as finding tracks that go through these potential locations which is also known as data association and is typically formulated as an optimization problem.

[52] formulates data association as an integer linear program, which is solved via a relaxation to linear programming. [42] formulates data association as a linear assignment problem, which is solved via the classic Hungarian algorithm. In order to deal with occlusion, heuristics are used to hypothesize potential merging and splitting of targets, which are incorporated into the optimization problem. Postprocessing is then used to recover object identities across hypothesized splits and merges. [60] couples the problem of multi-object tracking with object detection and formulates a quadratic boolean program. [110] introduces a network flow formulation where observation and transition likelihood are encoded as flow cost and is solved with the min-cost flow algorithm. An iterative approach is built to deal with miss detection where in each iteration, occluded object hypotheses are generated according to the previous solution. [15] formulates tracking as a maximum weight independent set problem. Small tracklets are iteratively merged into longer ones to handle long-term occlusions. [1] improves the network flow by maintaining identity for different flows using multi-commodity flow. As the problem is NP-hard, relaxation is required.

These approaches all make certain modeling simplifications, such as a first-order Markovian dependency structure, which has the benefit of efficient solutions methods that are sometimes exact. These methods, however, typically require post-processing or an iterative process to refine the solution, which are usually heuristic and non-recoverable when errors occur. These methods are also, almost exclusively, based on linking detections, which can cause them to break down when detectors are unreliable.
2.3 TRACKING-BY-SEGMENTATION

The above approaches all use bounding-box based representations and thus suffer from the problems mentioned in the introduction. A few efforts have been done in tracking-by-segmentation [11, 31, 38, 98]. One approach [98] learns a discriminative appearance model to help identify whether a superpixel belongs to the target or background. Another approach follows a particle filtering framework for traditional tracking and then uses segmentation to segment each particle sample [11]. More recently, an approach based on online learning of boosted decision trees was used to classify patches and the target segment is computed by thresholding the confidence map followed by morphological operations [91]. In other work [99], the target is represented by a multiple parts model and an optimization framework is proposed to jointly label superpixels as one of the target parts or as background and at the same time update the part model. Similarly, [106] formulates an energy minimization problem to solve tracking and segmentation jointly where the objective contains a tracking term, a segmentation term and a coupling term that forces consistency between the tracking and segmentation solutions. Dual decomposition is used to iteratively solve the problem. While these methods have shown promising empirical performance, they typically require relatively long runtimes for all of the components and/or have a non-trivial number of hyperparameters associated with the set of components that must be carefully tuned.

2.4 DEEP NETWORKS FOR TRACKING

Deep networks have become a dominant technique in computer vision and have been applied to almost all vision problems. In tracking, one approach [97] trains an autoencoder offline to learn generic image features which are then transferred for online learning of target appearance during tracking. A similar but more complex approach [96] regards a pretrained CNN as an ensemble with each channel of the output feature map as an individual base learner. During online training, base learners are updated and sequentially sampled into the ensemble set. An alternative approach [65] uses the convolutional feature maps from several layers of a pretrained CNN and applies correlation filters hierarchically on these maps to find the target location. A siamese network has been used for tracking [94] in order to learn a similarity function between a target template and candidate bounding boxes. All of these approaches focus on estimating the location and scale of the target via a predicted bounding box but cannot provide a precise pixel-level target mask. [32] uses the particle filtering framework where each particle is evaluated via
a deep network. The network uses several RNNs to explore the inner structure of the target. Each RNN explores the image according to a particular direction. In [111], a triplet network is designed to learn discriminative features. The network takes three images as input and each image goes through the same CNN branch. During training, each training sample consists of a positive pair where the two images belong to the same target and a negative pair. The network is trained such that in the embedded space, the distance between the positive pair is smaller than the negative pair. [72] maintain a tree of CNN models of the target online. Each CNN model in the tree comes from the original network finetuned on a set of frames online. Each CNN model has a reliability score that is estimated using the path in the tree up to this model. The score is recursively updated by the edge affinity between two CNN nodes, where the edge affinity is evaluated by how well a node performs in terms of the training samples of the other model. A target candidate is evaluated as the weighted sum of the scores from all CNN models in the tree where the weight comes from the reliability of each model.
Chapter 3: Over-Segmentation to Tracking

3.1 INTRODUCTION

In this work, we present a new approach to tracking multiple interacting people in real-world videos, where detecting people occurrences is not reliable. Examples include videos of pedestrians in crowded scenes, and team sports such as basketball and volleyball. In these videos, people are subject to long-term partial or full occlusions. Also, in sports videos, players may assume a wide range of poses and body articulations, non-linear motion profiles, and players on the same team have very similar appearances.

These challenges make the state-of-the-art people detectors unreliable. As illustrated in Figure 3.1, our experiments demonstrate that responses of the popular DPM detector [34] are very noisy, leading to poor performance of state-of-the-art trackers based on data association of detections. This raises two key questions that motivate our work.

1. What video features could be more suitable for grounding data association in our setting compared to detections?

2. Do popular, state-of-the-art data-association frameworks for detection-based tracking still work for these features, or are new methods required?

Below we address each of these questions and overview the resulting framework proposed in this work.

3.1.1 VIDEO FEATURES

While the performance of state-of-the-art people detectors has continually improved, they still perform poorly in certain challenging situations. As shown in Figure 3.1, one typical problem is occlusion. Detectors usually miss people under partial occlusion either because they are incapable of dealing with arbitrary sets of missing parts or because of post-processing steps such as non-maxima suppression. Another problem encountered by standard people detectors is that they are unable to robustly handle large pose variation which happens frequently in sports.
videos. A potential avenue for dealing with these issues is to train specialized detectors on possible pose/occlusion variations in the domain of interest. While this is worth investigating, the combinatorial space of possible variations makes the practicality of such an approach questionable. Further, as the number of distinct detectors grows, then so will the overall false positive rate of the system. All of these issues raise additional challenges for data association, which must handle merging and splitting of detections [42], as well as hypothesizing missing detections [110].

In this work, we argue that mid-level temporal features derived from video over-segmentation are more suitable for our target applications. In particular, rather than assuming reliable people detections, we assume that our videos can be reliably over-segmented in space-time, such that segments typically do not cross the boundaries of people being tracked. The segments then serve as the temporal mid-level features and our basis for tracking. Unlike detections, boundaries of these over-segmented features, e.g. supervoxels, typically align with people’s contours. Therefore, tracking the right combination of mid-level features facilitates bottom-up reasoning under occlusion and varying poses. As Figure 3.1 shows, this can help disambiguate multiple partially occluding individuals (top right) and easily delineate individuals in non-standard poses (bottom right). This would also allow us to relax the strong assumptions about the motions, sizes, and shapes of the tracked objects, typically made by prior work on detection-based tracking.

3.1.2 DATA ASSOCIATION

Our choice of mid-level temporal features as a basis for tracking presents fundamental challenges to the common network-flow and related formulations of data association, which represent the state-of-the-art methods for detection-based tracking (e.g., [79, 64, 6, 102, 104, 90, 60]). In particular, such approaches largely assume that individuals in a frame should be associated with a single detection/feature. This assumption, however, is violated when using mid-level temporal features since a person being tracked is typically represented by an unknown number of mid-level features, which split and merge in space-time.

One might still consider formulating the tracking of such mid-level features within a network-flow framework. This, however, poses several fundamental challenges. One of the key challenges is setting the “edge capacities” of the network flow optimization problem. Most approaches set edge capacities to be uniformly one, which indicates that one flow represents one person track. As mentioned above, this is generally not the case with mid-level features and the
ideal setting for edge capacities is unclear.

One approach to applying efficient network flow solvers might be to heuristically hypothesize where splits and merges of mid-level features occur in order to set edge capacities. However, this heuristic simplification can easily lead to an unrecoverable loss in accuracy, negating the advantage of using efficient “optimal” solvers. Further, the types of constraints and affinities among mid-level features that can be represented in such Markovian flow networks is quite limited. Our initial efforts toward this style of approach have not yet been successful.

Another approach would be to use extensions to the standard network-flow paradigm, such as multi-commodity flow networks [1]. Unfortunately, these extensions typically result in computational hard optimization problems that require approximate solutions. Further, these extensions typically inherit the Markovian restrictions on constraints and affinities from the standard flow paradigm.

### 3.1.3 OUR APPROACH: CSL

The above considerations about flow-based data association have led us to consider a fundamentally different approach. In this work, rather than heuristically simplify the mid-level labeling problem to an efficiently solvable framework, we instead develop a greedy labeling approach that can operate directly on the original (un simplified) problem. In particular, our approach conducts tracking by sequentially labeling mid-level video features with object identifiers, under hard constraints. Our new algorithm, called constrained sequential labeling (CSL), uses a flexible cost function to sequentially assign labels while directly respecting the implications of hard constraints (e.g., a person cannot be at two distinct locations). A key advantage of this approach is that it places few restrictions on the form of the cost function, allowing it to capture higher-order dependencies among mid-level features.

Our formulation of tracking as labeling mid-level features can be viewed as a structured prediction problem, which has received considerable attention. Our CSL solution approach is inspired by the success of sequential classification methods for structured prediction problems, ranging from natural language processing to control, e.g. [28, 40, 83]. The idea is to learn a classifier for making a sequence of labeling decisions (e.g. labeling mid-level features) resulting in a final output. The classifiers are sequential in the sense that their predictions can depend on previous predictions in the sequence.

Importantly, while our approach is inherently greedy, it comes with strong theoretical guar-
antees. We prove that if there is a cost function (within the considered cost function space) that supports accurate labeling given the constraints, then the learning algorithm will achieve finite-time convergence to such a cost function, where the rate of convergence improves with the “strength” of the constraints. As our experiments demonstrate these assumptions are satisfied in challenging real-world problems, where we are able to learn high-quality constraints and cost functions that support accurate CSL. In particular, we present experiments using sports and pedestrian data that demonstrate significant improvements over the state-of-the-art.

Overall the main contributions of this work are summarized as follows:

• The CSL framework for tracking based on mid-level features;
• Learning algorithms for training the cost function and constraints, used by CSL;
• Proof that our learning algorithms converge and the rate improves with the strength of constraints;
• Empirical evaluation that demonstrates the effectiveness of CSL for labeling mid-level features relative to network-flow based data association.

We present this work as follows. In Section 3.2, we formulate the problem setup and describe the overall constrained sequential labeling approach. Section 3.3 then specifies the features used to represent our cost functions and the learning algorithm used to learn the cost function parameters. Our representation and learning algorithm for constraints is then presented in Section 3.4 followed by a presentation of our experimental results in Section 3.5. Finally, we summarize and discuss future work.

3.2 CONSTRAINED SEQUENTIAL LABELING

Our input for multi-object tracking is a set of \( n \) mid-level, temporal features \( S = \{S_1, S_2, \ldots, S_n\} \) (e.g. supervoxels), which represent an over-segmentation of the foreground object activity in a video (e.g. as determined via background subtraction, saliency detection or object detection). Given \( S \), the desired output is to label each \( S_i \) by a label \( l_i \in \{1, \ldots, k\} \), which indicates which of \( k \) objects \( S_i \) corresponds to. For simplicity, we first assume that \( k \) is known and drop this assumption later in Section 3.2.4. Our approach is designed with two additional assumptions in mind: 1) Visible target objects in the initial frame are labeled (manually or by a detector), noting that we do allow for non-visible target objects, and 2) The mid-level features in \( S \) are at a
Figure 3.1: Bounding boxes of the DPM detector [34] (left), and supervoxels of [103] (right). The DPM often misses players in crowds (top) and in unexpected poses (bottom). Supervoxels are more suitable for our challenging videos, because their boundaries are usually aligned with part boundaries facilitating bottom up reasoning.
fine enough resolution so that they typically respect the space-time extent of objects, rather than commonly crossing object boundaries. We now describe the overall structure of our Constrained Sequential Labeling (CSL) algorithm and then describe each of its components in more detail.

3.2.1 MAIN ALGORITHM

CSL operates by iteratively assigning labels to features and removing labels from consideration. For this purpose, the algorithm keeps track of the assigned and removed labels via a partial labeling $L$, which is a mapping from features to label sets. In particular, $L(S_i) \subseteq \{1, \ldots, k\}$ represents the set of labels that have not been ruled out for feature $S_i$ at a particular point in the algorithm. If $L(S_i) = \{l\}$ then $S_i$ is said to be assigned label $l$. In cases where $|L(S_i)| > 1$, we say that $L$ specifies a partial label for $S_i$, meaning that the algorithm has not yet made a final label determination.

CSL initializes the partial labeling such that $L(S_i) = \{1, \ldots, k\}$ if $S_i$ does not start from the first frame (i.e. all labels are possible), and otherwise $L(S_i) = \{l\}$, where $l$ is the label specified for $S_i$ in the first frame. After initialization, CSL iteratively refines $L$ by removing one or more labels from the partial labeling of one or more features. The refinement continues until reaching a fully specified labeling of $S$. The knowledge used to make each refinement decision is captured by both a cost function and inequality constraints, which are applied in two distinct steps. First, inequality constraints are used to propagate label information to other features, which can have the effect of reducing the number of feasible labels in the partial labeling. In the second step of each iteration, the cost function is used to select and label a single feature based on its affinity with already labeled feature groups. The iteration continues until either all features are labeled or no feature can be labeled without violating an inequality constraint. We describe this process below, first ignoring constraints and then incorporating constraints. Algorithm 20 gives pseudo-code for CSL. Note that lines 6-11 in Algorithm 20 pertain to learning, explained in Section 3.3.

3.2.2 SEQUENTIAL LABELING WITH A COST FUNCTION

We now describe sequential labeling (SL) using a cost function and no constraints. We assume a cost function $C(S_i, l \mid L)$ that assigns a cost to the decision of labeling $S_i$ as $l \in L(S_i)$, where $L$ is the current partial labeling being refined. Given the current partial labeling $L$, each iteration
Algorithm 1 Constrained Sequential Labeling Algorithm with learning option. When the flag learn is set, the algorithm performs one run of learning the weight vector $w$.

**Input:** A set of mid-level features $\{S_i\}$
- Object labels for first frame
- Weight Vector $w$
- Set of inequality constraints $\Sigma$
- Learning Option learn with ground truth labeling $L_{gt}$

**Output:** A labeling $L$ or weight vector $w$

1. Initialize partial labeling $L$ according to label information in first frame
2. while there exists $S_i$ such that $|L(S_i)| > 1$ do
3. $L = \text{ConstraintPropagation}(L, \Sigma)$
4. $D = \{(S, l) : S \in A(L), l \in L(S)\}$
5. $(S_i, l) = \arg\min_{(S, l) \in D} C(S, l | L)$
6. if learn and $L_{gt}(S_i) \neq l$ then
7. $C^+ = \{(S, l) \in D | L_{gt}(S) = \{l\}\}$
8. $(S^+, l^+) = \arg\min_{(S, l) \in C^+} C(S, l | L)$
9. $C^- = \{(S, l) \in D - C^+ | C(S, l | L) < C(S^+, l^+ | L)\}$
10. $w = w + \sum_{(S, l) \in C^-} \frac{\Psi(S, l, L)}{|C^-|} - \sum_{(S, l) \in C^+} \frac{\Psi(S, l, L)}{|C^+|}$
11. $L(S^+) = \{l^+\}$
12. else
13. $L(S_i) = \{l\}$
14. end if
15. end while
16. if learn then
17. Return $w$
18. else
19. Return $L$
20. end if
begins by considering a set of active features (see below) denoted by $\mathcal{A}(L)$ as possibilities for assigning a new label. The cost function $C$ is then used to select an active feature $S_i \in \mathcal{A}(L)$ and a corresponding label $l \in L(S_i)$ to assign to it by setting $L(S_i)$ to $\{l\}$. In particular, the pair $(S_i, l)$ is selected to be the pair with lowest cost considering all active features and labels for those features. These steps are depicted in lines 4, 5, and 13 of the pseudo-code.

Intuitively, we want $C$ to assign the lowest costs to correct pairs that can be most confidently labeled in the context of $L$. In this work, we will use linear cost functions that are expressed as $C(S_i, l \mid L) = w \cdot \Psi(S_i, l, L)$, where $w$ is an $m$-dimensional weight vector and $\Psi(S_i, l, L)$ returns an $m$-dimensional descriptor vector that represents information about the corresponding decision. The descriptors used in this work are presented in Section 3.3.1 and Section 3.3 gives a learning algorithm for $w$.

It remains to specify the active features $\mathcal{A}(L)$. Given a current $L$, $S_i \in \mathcal{A}(L)$ (i.e. is active) if its label is partial and is temporally close (within 5 frames) to an already labeled feature (see Figure 3.2). Restricting our attention to active features reduces the number of cost function evaluations required at each evaluation in order to select the best feature-label pair, which can result in significant speedups. This particular definition of $\mathcal{A}(L)$ is intuitively sensible since generally the cost function is most confident about features that are nearby already labeled ones. This definition of $\mathcal{A}(L)$ is also quite flexible, allowing for qualitatively different labeling behaviors depending on the cost function. For example, the SL process might start by assigning all labels to a single object from the start to end of a video, or it might alternate between different objects in temporal order.

### 3.2.3 INEQUALITY CONSTRAINTS AND PROPAGATION

Using the cost function alone for SL can be effective for simpler tracking problems, for example, ones that do not involve significant occlusion. However, for more difficult problems, we have found (see experiments) that it is difficult to learn or hand-code cost functions that achieve state-of-the-art performance. This is due to the short-sighted, greedy nature of pure SL. Fortunately we have found that this poor performance can be overcome by combining greedy SL with constraint reasoning, which lead to the CSL algorithm. Section 3.3 provides a theoretical characterization of this observation.

Our CSL approach uses a set of inequality constraints $\Sigma$ associated with each video. Each constraint is of the form $S_i \neq S_j$, indicating that the features cannot be assigned the same label.
For example, it will be common for inequality constraints to hold for features \( S_i \) and \( S_j \) that are temporally close but spatially far apart, which indicates that they are likely not from the same object. A partial labeling \( L \) violates a constraint \( S_i \neq S_j \) if \( L(S_i) = \{l\} \) and \( l \in L(S_j) \) or vice versa. That is, if one of the features is assigned a single label, then that label cannot be a possibility for the other feature. We say that \( L \) is consistent with \( \Sigma \) if it is not violated by any constraint. Below we describe how CSL uses constraints. Later, in Section 3.4, we will describe how we learn constraints that are nearly always valid.

Perhaps the simplest way to combine a constraint set \( \Sigma \) and a cost function is to limit the labeling choices of the cost function to ones that will not result in constraint violations given the current partial labeling \( L \). For example, if we have a constraint \( S_i \neq S_j \) and our current labeling has \( L(S_i) = \{l\} \), then when evaluating \( S_j \) we need not consider the cost of \( l \) as it has been eliminated from consideration. Note that this way of using constraints would be equivalent to defining a cost function that effectively assigns an infinite cost to \( (S_i, l) \) pairs that violate a constraint given the current \( L \).

While using constraints as described above can improve efficiency by reducing the number of cost function evaluations, our results show that the improvement in accuracy over pure SL is modest. The weakness of this approach to incorporating constraints is that it only considers
immediate constraint violations, rather than further reaching implications of the constraints. For example, suppose that currently $L(S_i) = \{1\}$ and $L(S_j) = \{1, 2\}$, and we have the constraint $S_i \neq S_j$. We can infer that the label for $S_j$ is 2 since it cannot be 1 and given this new information we can eliminate label 2 as a possibility from any feature that has an inequality constraint with $S_j$, and so on. This process of iteratively computing the implicit consequences of constraints is known as constraint propagation [27] and is an extremely important component of state-of-the-art constraint satisfaction and optimization algorithms.

More precisely, constraint propagation is a procedure that operates on a current partial labeling $L$ and a constraint set $\Sigma$. We let ConstraintPropagate$(L, \Sigma)$ denote the constraint propagation procedure, which returns a refined partial labeling $L'$ that is consistent with $\Sigma$. Specifically, $L'$ will be the minimal refinement of $L$ (maximally close) that is consistent. ConstraintPropagate$(L, \Sigma)$ is straightforward to implement for inequality constraints and corresponds to the unit propagation algorithm commonly employed in the area of constraint satisfaction [27]. Starting with $L$ iteratively finds a violated constraint $S_i \neq S_j$ such that $L(S_i) = \{l\}$ and then removes label $l$ from $L(S_j)$ to get a new labeling. The process terminates when no violation can be found. This termination condition will always occur since the number of labels in $L$ is finite and each iteration removes one label from $L$.

CSL integrates constraint propagation into SL by calling the procedure at the beginning of each iteration (line 3). Thus, in the first iteration label information will be propagated from the known labels of the first frame. In later iterations, label information will be propagated in response to the labeling decision made by the cost function in the previous iteration. This propagation can rule out many possible labels that might have otherwise been considered in current or later iterations by the cost function. In this way, the constraints have the effect of improving efficiency since the number of cost function calls can be reduced since the labels removed by constraint propagation no longer need to be considered. At the same time, the accuracy can increase because some of the eliminated cost function calls may have resulted in erroneous label assignments. Our results show the improvements are significant.

### 3.2.4 HANDLING UNKNOWN NUMBERS OF OBJECTS

The above description of CSL assumed that the number of objects $k$ was known and fixed, which is the case in many sports domains, for example. However, CSL is easily modified to handle unknown $k$, which will occur in many other applications, such as pedestrian tracking. The above
CSL description is unchanged, except that $k$ is initialized to be the number of labeled objects in the first frame. During the iteration the label set is enlarged ($k$ is increased) whenever the results of constraint propagation remove all label possibilities for some feature $S$ (i.e. $L(S) = \emptyset$), indicating that $S$ should not be labeled as any of the current objects. Whenever this happens, a new object is added ($k$ is incremented) to the set of possible labels of any partially labeled features and $S$ is assigned the new label. When the learned constraints in $\Sigma$ are noisy, we may erroneously remove a correct label from $L(S)$ resulting in $L(S) = \emptyset$, which will hurt performance. Our experiments show that this is rare in some highly non-trivial domains and that the constraints provide an overwhelming positive impact.

3.3 COST FUNCTION REPRESENTATION AND LEARNING

In this section, we first describe the descriptors $\Psi(S_i, l, L)$ used to represent our linear cost function $C(S_i, l \mid L) = w \cdot \Psi(S_i, l, L)$. We then describe the algorithm used to learn the cost function weights $w$ and prove conditions for its convergence.

3.3.1 COST FUNCTION DESCRIPTORS

Given a partial labeling $L$, the descriptor function $\Psi(S_i, l, L)$ is defined as:

$$\Psi(S_i, l, L) = [n_u, n_d, n_f, d_l, d_v, d_c]$$

The first type of descriptors $n_u, n_d$ and $n_f$ provide measures related to how confidently a feature can be labeled:

- $n_u$: a count of the number of features that are not labeled yet in the neighborhood of $S_i$ (features that are adjacent to $S_i$). Intuitively, our labeling confidence is related to the number of assigned labels that are nearby.

- $n_d$: a count of the number of features in the neighborhood of $S_i$ that have a different label than $l$. This allows for learning a preference for assigning labels that agree with neighboring labels.

- $n_f$: equal to $T - |S_i|$, where $T$ is the total number of frames and $|S_i|$ is the number of frames the feature spans. This allows for learning a preference (or cost) for labeling longer
features before shorter ones.

Overall this set of features allows the system to learn preferences for labeling temporally long features with many labeled neighbors that have the same assigned label.

The second set of descriptors $d_l$, $d_v$ and $d_c$ (formalized below) encodes the usual visual dissimilarity between a feature and an object template in terms of location, optical flow [33], and color histograms (HSV). The object template for label $l$ is the union of all $S_i$ such that $L(S_i) = \{l\}$. We update the corresponding object template after each CSL iteration. Note that by introducing the object template, the cost function becomes extremely high order compared to most previous work where only lower order relations are considered.

For location and optical flow, the template is maintained every frame, and dissimilarity is calculated by averaging over the Euclidean distance between the feature and the template across each overlap frame. For color histogram we keep one model and the dissimilarity is directly the distance between the feature and the template. In order to account for the differences of color and motion among parts of an object, the template for color and optical flow is clustered into 3 parts via k-means. Given a feature and the template, the dissimilarity for color and motion is calculated according to the closest cluster. A detailed description of these descriptors is given below.

$$d_l = \begin{cases} \frac{1}{\min(t_f^l, t^{|S_i|}) - t_0^l} \sum_{t=t_0^l}^{\min(t_f^l, t^{|S_i|})} d(\text{loc}(s^l_t), \text{loc}(T^l_t)) & \text{if } t_0^l \leq t_f^l \\ d(\text{loc}(s^0_t), \text{loc}(T^l_{t_f^l})) & \text{otherwise} \end{cases}$$

where $t_f^j$ denotes the $j$th frame of feature $S_i$, $t_f^l$ denotes the current last frame of template $l$, $s^l_t$ is the part of feature $S_i$ that intersects with frame $t$, $d(\cdot)$ calculates the Euclidean distance, $T^l_t$ is the template for label $l$ at frame $t$, and $\text{loc}(\cdot)$ is the location of a feature or a template in a certain frame.
\[ d_v = \begin{cases} \frac{1}{\min(t^e_i, t_i^{\inf}) - 1} \sum_{t=t^0_i}^{\min(t^e_i, t_i^{\inf}) - 1} d(\text{opt}(s^i_t), \text{opt}(\hat{T}^i_t)) & \text{if } t^0_i \leq t^e_i \\ d(\text{opt}(s^0_i), \text{opt}(\hat{T}^i_{t-1})) & \text{otherwise} \end{cases} \]  

(3.2)

where \( \text{opt}(\cdot) \) calculates the histogram of optical flow of a feature or a template. Here \( \hat{T}^i_t \) is the cluster that the feature belongs to and is formally defined as:

\[ \hat{T}^i_t = \arg \min_{T^i_{t,k}} d(\text{opt}(s^i_t), \text{opt}(T^i_{t,k})) \]  

(3.3)

where \( T^i_{t,k} \) enumerates over different clusters for template \( l \) at frame \( t \).

\[ d_c = d(\text{color}(S_i), \text{color}(\hat{T}_t)) \]  

(3.4)

Here \( \text{color}(S_i) \) returns the HSV color histogram and \( \hat{T}_t \) denotes the corresponding cluster.

### 3.3.2 COST FUNCTION LEARNING

We now describe a learning algorithm for tuning the weights of our linear cost function \( C(S_i, l \mid L) = w \cdot \Psi(S_i, l, L) \) and give conditions for its convergence. For the purposes of supervised learning, we assume a set of labeled training videos \( \{(S^{(i)}, L^{(i)}, \Sigma^{(i)})\} \), where \( S^{(i)} \) is the set of features for the \( i \)'th video, \( L^{(i)} \) gives the ground truth label assignments for \( S^{(i)} \), and \( \Sigma^{(i)} \) is the set of constraints for \( S^{(i)} \). Note that the constraints will be learned by an independent process described in Section 3.4. The goal is to learn a set of weights such that CSL achieves high accuracy on each training video in the context of the constraints for those videos.

This learning problem poses at least three challenges over standard supervised learning. First, the training data is ambiguous since it does not indicate the exact sequence of labeling decisions. At any iteration of CSL there are many active features that could be chosen for labeling, however, the training data only tells us the labels of those features, but not which one the cost function should select for labeling at each step. The second challenge is that the sequence of decisions made by the cost function are not independent since each decision depends on the current partial labeling \( L \), which was generated based on previous decisions. The third challenge...
is that learning should be done in the context of the constraints and thus the cost function is not necessarily responsible for making all labeling decisions. Rather, constraint propagation will often assign labels to features. The cost function should be learned in a way that is synergistic with the information provided by constraints.

Our algorithm, CSL-Learn, addresses these challenges by directly integrating learning into CSL. We follow an online learning approach, where the algorithm repeatedly iterates through the training videos processing one video at a time. At a high level, the processing of one video may result in multiple updates to \( w \) in response to observed errors using the current weight vector. The iteration through training videos ends when either a specified number of iterations is reached or accuracy is perfect. It remains to describe the details of how each training video is processed during learning.

For each training video, the learning algorithm executes a “learning enhanced” CSL procedure given by Algorithm 20 with the flag, \texttt{learn}, turned on. The algorithm behaves like CSL until the cost function using \( w \) suggests an incorrect labeling decision (line 6), where the correctness is judged relative to the ground truth labeling \( L_{gt} \). At this point the weight vector is updated in a way that discourages errors at similar decision points in the future (see below). In addition, the incorrect decision is not used to update the partial labeling, but rather the correct labeling with the lowest cost decision is used to update \( L \) (line 11). The CSL process then continues as normal until another labeling error is observed, which results in another weight update.

It remains to specify the weight vector update when an incorrect label \( l \) is assigned to feature \( S_i \) given partial labeling \( L \). We define \( C^+ \) to be the set of all correct labeling decisions involving active features (i.e. features in \( A(L) \)) and \((S^+, l^+)\) to be the least cost decision in \( C^+ \) according to the current weight vector. We also define \( C^- \) to be all incorrect labeling pairs that have a smaller cost than \((S^+, l^+)\), i.e. the incorrect pairs that are preferred over all correct pairs. The weight vector is updated using the following Perceptron-style rule:

\[
    w = w + \sum_{(S, l) \in C^-} \frac{\Psi(S, l, L)}{|C^-|} - \sum_{(S, l) \in C^+} \frac{\Psi(S, l, L)}{|C^+|} \tag{3.5}
\]

This rule updates \( w \) so as to increase the average cost of all pairs in \( C^- \) (high-ranking incorrect pairs) and decrease the average cost of the correct pairs in \( C^+ \). The intuition is that this will lead to at least one correct decision to outrank all incorrect decisions as the iterations proceed.
3.3.3 CONVERGENCE OF LEARNING

We now consider convergence conditions for the above learning algorithm, which shed some additional light on the potential benefits of integrating constraints into sequential labeling. We analyze the realizable learning setting, where it is possible to find a weight vector that correctly labels all of the training data. In particular, as for most Perceptron-style learning algorithms, we bound the number of mistakes made during the training process until the weight vector correctly labels all training data. We must assume here that the inequality constraints for each training example are consistent with the ground truth labeling, since otherwise convergence would not be possible.

As for the classic Perceptron algorithm, our convergence guarantee is in terms of a notion of margin. Given a set of training videos, let \( \mathcal{L} \) be the set of partial labelings that are consistent with the inequality constraints. Given a constraint set \( \Sigma \), the \( \Sigma \)-constrained margin of a weight vector \( w \) for a training set is the minimum over all \( L \in \mathcal{L} \) of \( w \cdot \Psi(S^-, l^-, L) - w \cdot \Psi(S^+, l^+, L) \), where \( (S^+, l^+) \) is any correct candidate labeling and \( (S^-, l^-) \) is any incorrect candidate labeling. If \( w \) has a positive constrained margin then using it within CSL with the constraints will correctly label the training set. We now show that the existence of a positive margin \( w \) is sufficient for convergence. Below \( R \) is a constant such that for all possible feature vectors \( \Psi(S, l, L) \) and \( \Psi(S', l', L) \) we have \( \| \Psi(S, l, L) - \Psi(S', l', L) \| \leq R \).

**Theorem 1** Given constraints \( \Sigma \) and any training set such that there exists a weight vector \( w \) with \( \Sigma \)-constrained margin \( \gamma > 0 \) and \( \| w \| = 1 \), CSL-Learn will converge to a weight vector that correctly labels all training examples after making no more than \( \left( \frac{R}{\gamma} \right)^2 \) weight updates.

Let \( w^k \) be the weight vector before the \( k \)th update. We first derive an upper bound on \( \| w^{k+1} \| \).

\[
\| w^{k+1} \|^2 = (\| w^k \| + \| \sum_{(S,l) \in C^-} \frac{\Psi(S,l,L)}{|C^-|} - \sum_{(S,l) \in C^+} \frac{\Psi(S,l,L)}{|C^+|} \|)^2 \\
= \| w^k \|^2 + \| \sum_{(S,l) \in C^-} \frac{\Psi(S,l,L)}{|C^-|} - \sum_{(S,l) \in C^+} \frac{\Psi(S,l,L)}{|C^+|} \|^2 \\
+ 2w^k \cdot ( \sum_{(S,l) \in C^-} \frac{\Psi(S,l,L)}{|C^-|} - \sum_{(S,l) \in C^+} \frac{\Psi(S,l,L)}{|C^+|} ) \\
\leq \| w^k \|^2 + \| \sum_{(S,l) \in C^-} \frac{\Psi(S,l,L)}{|C^-|} - \sum_{(S,l) \in C^+} \frac{\Psi(S,l,L)}{|C^+|} \|^2
\]
The first inequality holds because the $C^-$ is defined such that all pairs in $C^-$ have lower cost than any pair in $C^+$ given the current weight vector $w^k$, which makes the cross term negative.

The second inequality follows directly from the definition of $R$. From the above equation, we can use induction to derive that $\|w^{k+1}\|^2 \leq kR^2$ (note that $w^1 = 0$).

We now derive a lower bound on $w \cdot w^{k+1}$.

\[
w \cdot w^{k+1} = w \cdot w^k + w \cdot \left( \sum_{(S,l) \in C^-} \frac{\Psi(S,l,L)}{|C^-|} - \sum_{(S,l) \in C^+} \frac{\Psi(S,l,L)}{|C^+|} \right)
\]

\[
= w \cdot w^k + \sum_{(S,l) \in C^-} \frac{w \cdot \Psi(S,l,L)}{|C^-|} - \sum_{(S,l) \in C^+} \frac{w \cdot \Psi(S,l,L)}{|C^+|}
\]

\[
\geq w \cdot w^k + \gamma
\]

The inequality comes from the definition of $\gamma$. Using induction we can now derive $w \cdot w^{k+1} \geq k\gamma$. Combining this lower bound with the above upper bound we get the following.

\[
1 \geq \frac{w \cdot w^{k+1}}{\|w\|\|w^{k+1}\|} \geq \sqrt{k} \frac{\gamma}{R}
\]

From this and noting $\|w\| = 1$ we get the final result, $k \leq (R/\gamma)^2$ The above result shows that a larger margin implies a better mistake bound. We now relate the constraints to margins.

**Proposition 1** For any constraint sets $\Sigma$ and $\Sigma'$, such that $\Sigma \subseteq \Sigma'$, the $\Sigma'$-constrained margin of any weight vector $w$ is no less than the $\Sigma$-constrained margin. Furthermore there exists a training set and constraint sets $\Sigma \subseteq \Sigma'$ such that there is a weight vector $w$ with positive $\Sigma'$-constrained margin and there is no weight vector with positive $\Sigma$-constrained margin.

We denote $L_\Sigma^c$ as the set of all valid candidate feature label pairs under constraint set $\Sigma$. With $\Sigma \subseteq \Sigma'$, we then have $L_{\Sigma'}^c \subseteq L_\Sigma^c$, since more constraints will only make the valid candidate set smaller. Next, consider the definition of $\gamma$ again:

\[
\gamma = \min w \cdot \Psi(S^-,l^-,L) - w \cdot \Psi(S^+,l^+,L)
\]
Since the selection of \((S^+, l^+), (S^-, l^-)\) are independent, \(\gamma\) can be further defined as the difference between the minimum of \(w \cdot \Psi(S^-, l^-, L)\) and the maximum of \(w \cdot \Psi(S^+, l^+, L)\). For a particular dataset, all \((S^+, l^+)\) pairs are fixed with respect to the constraint set since we assume the constraint set doesn’t conflict with the ground truth labeling. Thus given a different constraint set, only the first term will be influenced.

\[
\gamma_{\Sigma} - \gamma_{\Sigma'} = \min_{(S^-, l^-) \in L_{\Sigma}^c} w \cdot \Psi(S^-, l^-, L) - \min_{(S^-, l^-) \in L_{\Sigma'}^c} w \cdot \Psi(S^-, l^-, L)
\]

Since, we have \(L_{\Sigma'}^c \subseteq L_{\Sigma}^c\), then

\[
\min_{(S^-, l^-) \in L_{\Sigma}^c} w \cdot \Psi(S^-, l^-, L) \leq \min_{(S^-, l^-) \in L_{\Sigma'}^c} w \cdot \Psi(S^-, l^-, L)
\]

so \(\gamma_{\Sigma} \leq \gamma_{\Sigma'}\). The next statement follows directly. Since \(\gamma_{\Sigma} \leq \gamma_{\Sigma'}\), there should exist a dataset that makes \(\gamma_{\Sigma} = 0\), then \(\gamma_{\Sigma'} \geq 0\). So there exists a weight vector that makes \(\gamma_{\Sigma'} > 0\). Combining the above results we can see the utility of using constraints in learning. First, adding constraints will never decrease the margin and often increase it, which means the mistake bound will never get worse and often improve, suggesting learning is made easier. Second, there are problems where constraints are necessary to obtain a positive margin and hence guarantee convergence.

### 3.4 CONSTRAINT LEARNING

Recall that CSL operates on a set of constraints that are specific to each video. In order to produce, such constraints, we now consider how to learn a constraint generator that returns the inequality constraints over mid-level features in any target video. The generated constraints should have high precision in the sense that if \(S_i \neq S_j\) is generated, then it agrees with the ground truth with high probability. Further, we would like the constraint set to be as large as possible, while maintaining high precision, in order to maximize constraint propagation.

Our constraint generation is similar in spirit to prior work that used constraints between point trajectories [36]. In that work, the connected components of each video frame were computed and an inequality constraint was included between trajectories if they belonged to different components. We found that for our features that approach often produces erroneous constraints because it is not unusual in our videos for a single object to span multiple connected components.
in a frame. This occurs, for example, due to imperfections in our foreground saliency mask.

Fortunately, it is often still the case that when an object does span multiple components, those components will satisfy certain spatial layout properties. For example, for people tracking, if we consider a bounding polygon for two components, certain types of bounding polygons are unlikely to correspond to a single individual. Based on this insight, we train an SVM classifier to predict whether two components in a frame correspond to different objects. To do this we use labeled training data to extract pairs of components across the video frames and label them as positive if the components correspond to different objects and otherwise assign a negative label. We then compute features of the pairs and train the SVM classifier. In our experiments we found that a simple set of four features were sufficient for achieving good results. The features include: the horizontal and vertical distance between component centers and the height and width of the minimal bounding box containing the components. Given the learned SVM classifier, we then adjust its prediction threshold so that it achieves high precision.

Given this classifier we then generate a feature inequality constraint \( S_i \neq S_j \) if in some video frame \( S_i \) and \( S_j \) are contained in components that the SVM judges are from different objects. The pseudo-code for generating constraints are shown in Alg.2. On our test data, this approach can achieve a precision of over 97% in challenging volleyball videos while maintaining a recall of over 44%, which gives very informative constraint sets. Figure 3.3 illustrates an example of the constraint generation process.

3.5 EXPERIMENTAL RESULTS

Datasets. We evaluate using benchmark datasets of volleyball and basketball videos that involve the challenges noted in the introduction. In particular, standard people detectors are quite noisy in these domains. Also to compare to state-of-the-art detection-based approach, we report results on the widely used pedestrian benchmark video PETS2009-S2L1, S2L2 [35] where people detectors are effective. The Volleyball dataset [4] contains 38 videos of entire collegiate volleyball plays (each 200–800 frames, \( 720 \times 480 \)). Volleyball videos are very challenging, because of inter-occlusions of many players in different body articulations, and non-smooth fast motions. Ground truth bounding boxes are provided for the six near-team players, while opposing players are not annotated or used for evaluation. Specifically, only six players of the team in the front are annotated, whereas the players of the opposing team in the back are not of interest. We extended the ground truth to include finer-level annotations every 5 frames in terms of player pixel masks.
Algorithm 2 Inequality constraint generation.

Input: A set of mid-level features \( \{S_i\} \)
A set of foreground connected component \( \{C^t_a\} \)
A SVM classifier that outputs 1 for positive class

Output: A set of inequality constraints \( \Sigma \)

1: \( \Sigma = \emptyset \)
2: for \( t \leftarrow 1 \) to \( T \) do
3:   for all \( (C^t_a, C^t_b) \) pair in frame \( t \) do
4:     if SVM\( (C^t_a, C^t_b) \) == 1 then
5:       for \( S_i \in C^t_a \) and \( S_j \in C^t_b \) do
6:         if \( (S_i, S_j) \notin \Sigma \) then
7:           \( \Sigma = \Sigma \cup \{(S_i, S_j)\} \)
8:         end if
9:     end for
10: end if
11: end for
12: end for

which exactly delineate each player in these frames. We train and test via 3-fold cross-validation reporting averages across folds. The Figment dataset [36] contains 18 videos of basketball action (each 50–80 frames, \( 441 \times 180 \)). Ground truth player masks are provided every 7–8 frames.

Implementation Details. We tested CSL using both supervoxels (CSL-VOX) and dense point trajectories (CSL-DPT). Supervoxels were generated as the leaf-level supervoxels of the hierarchical video segmentation approach of [103]. Examples of supervoxels and dense point trajectories are shown in Fig. 3.4 The only input parameter for CSL-VOX is the total number of supervoxels, which is controlled by varying the video segmentation parameters. For generating dense point trajectories we followed [36]. For Volleyball we obtain foreground features via background subtraction based on a Gaussian-mixture color model. For the basketball videos, we follow [36] and extract foreground features based on motion saliency of dense point trajectories. For PETS2009-S2L1, we use a trained background model together with motion saliency to estimate the foreground features.

The initialization in the first frame can be done by either human annotated bounding boxes or detections. For the volleyball dataset, we use the ground truth bounding boxes in the first frame to initialize to label. To fairly compare with detection-based approach on PETS2009, we use detection outputs to initialize in the first frame rather than manual initialization.
Figure 3.3: Illustration of determining an inequality constraint, that is, whether two supervoxels can have the same label. In frame $t$, $S_i$ and $S_j$ belong to a single foreground connected component, so the frame does not yield an inequality constraint. In frame $t'$, the SVM classifier decides the two connected components cannot belong to a single individual, generating an inequality constraint between $S_i$ and $S_j$.

**Evaluation Metrics.** We use the standard CLEAR MOT evaluation metrics: miss detection (MD), false positives (FP), ID switches (ID-sw.), and accuracy (acc). Also, to compare to published results [36] on Figment we use the metrics from that work: per object clustering error (PRCE), recall, and tracking time.

**Baselines.** For Volleyball, we created baselines from common frameworks:

- **NCuts** [88] using different numbers of clusters 6 (the true number of clusters) and 12, and pairwise affinities required by the approach are computed in terms of color, space-time location, and optical flow of supervoxels. NCuts also uses the same responses of our inequality constraints classifier which are incorporated as zero-valued entries in the affinity matrix.

- Detection-based **Network Flow** [79] using the publicly available code that applies flow-based linking to people detections. We trained a people detector in our domain and used the resulting detections as input. We also updated the code to use appearance-based affinities, which improves results.
Network Flow*, the previous network flow approach, but applied to ground truth bounding boxes, rather than real detections. This is an oracle baseline (since perfect detections are used) intended to estimate idealized performance with perfect input.

- **Superpixel** uses our CSL approach on frame-based segments (intersections of supervoxels with frames), instead of temporally-extended supervoxels. This allows us to observe the importance of using temporally extended features within the CSL framework.

For the other two datasets, we compare against the reported state-of-the-art results.

**Quantitative Results.** Table 3.1 shows the Volleyball results. First, we observe that both CSL-VOX and CSL-DPT significantly outperform all non-Oracle approaches in all metrics. Further we see that CSL-VOX outperforms CSL-DPT by a small margin, primarily because the supervoxels provide more stable affinity estimation. The significant improvement over Superpixel shows the utility of using the more coherent temporal mid-level feature rather than frame-based features within CSL. Using the temporally extended features allows for more significant constraints and less shortsighted labeling. The comparison to Network Flow shows that a state-of-the-art detection based technique faces serious challenges in our domain, primarily due to the difficulty in obtaining accurate enough detections. Rather, by using mid-level features, the CSL approaches are less vulnerable to occlusions. Surprisingly, the CSL approaches are comparable
to the oracle Network Flow* approach, which is allowed to cheat and use ground truth detections. Notably, CSL is significantly better in terms of ID switches. Note that the oracle approach will necessarily achieve perfect MD and FP scores due to the use of perfect detections.

Table 3.4 shows that, on Figment, CSL outperforms two state-of-the-art methods [36, 16] based on graph partitioning of dense point trajectories with a variety of post-processing steps. We hypothesize that one reason for this is that the prior approaches relax hard constraints as affinity in the similarity matrix, which can result in non-sensical partitions that must be heuristically post-processed. CSL-VOX again outperforms CSL-DPT suggesting that supervoxels are a more effective mid-level feature in this domain.

Table 3.5 shows results of CSL and two top-performing detection-based methods on PETS2009S2L1. CSL achieves comparable performance, showing that even in domains where detections are more reliable, CSL is still competitive. The slightly worse precision performance of CSL is primarily due to the imperfect constraint classifier, which sometimes rules out all labels for certain supervoxels, which are then assigned new labels.

Table 3.5 shows results of CSL and two top-performing detection-based methods on PETS2009S2L1. CSL achieves comparable performance, showing that even in domains where detections are more reliable, CSL is still competitive. The slightly worse precision performance of CSL is primarily due to the imperfect constraint classifier, which sometimes rules out all labels for certain supervoxels, which are then assigned new labels.

There are not many results reported on S2L2. [71] achieves 46% in terms of MOTA. Our CSL-VOX achieves 53.8% here. The crowd density in this video is quite high. The main reason for the poor performance is that we can’t find a good set of supervoxels here. The supervoxels are either too large in the space domain that go across the object boundary or too short in time domain that make our constraints less powerful. This is an example video that significantly violates the assumptions of our approach, that is, we are not able to obtain features that form an over-segmentation of the video objects.

**Evaluation of Learning.** To evaluate the effectiveness of the proposed learning algorithm, we use a uniform weight vector. The results are shown as CSL-Uniform in Table 3.2. Surprisingly we see that the original version of CSL-VOX does not benefit much from learning. In particular, using a uniform weight vector was nearly as effective as using the learned weights. The reason for this was due to the fact that the features were carefully engineered to be on the same scale as one another in order to maximize their performance under uniform weighting.

In order to provide a better assessment of our learning algorithm, we ran experiments using the raw feature values before normalization. The results are also shown in Table 3.2. CSL-UnNormal uses the same set of features as CSL-VOX but the features are not normalized to the same scale. Further, the weight vector is set to be uniform. CSL-UnNormalTrain shows the results using unnormalized features after our learning algorithm. We can see that when the features are of different scales, our learning algorithm is able to find a good weight vector and
Table 3.1: VolleyBall dataset. Network Flow* is an oracle method that uses ground truth detections.

improve the results. This validates the effectiveness of our learning algorithm.

**CSL Variants.** We also evaluated variants of CSL to examine the importance of different components of CSL. The results are shown in Table 3.3.

We first exam the utility of the constraints. SL is pure sequential labeling (CSL without constraints) with weight learning, while CSL-noProp is CSL without constraint propagation, but does use constraints to assign infinite costs to labelings that immediately violate a constraint. We see that constraints and constraint propagation are crucial. In particular, SL (no constraints) is significantly worse than CSL. Further, CSL-noProp is able to use constraints to improve over SL, however, it is not nearly as effective as CSL, which uses propagation.

CSL-noColor and CSL-noFlow use the same CSL approach but without the color and optical flow features respectively. We can see that optical flow plays an very important role here while the color information is less useful. This is reasonable as in our videos, the players have similar uniforms.

Finally, we consider the performance of our constraint classifier in comparison to an alternative constraint classifier that uses raw optical flow, rather than the spatial layout of supervoxels. This provides an evaluation of the effectiveness of using motion for determining constraints versus spatial properties. Intuitively, for difficult sports videos, we expect that motion features will be quite noisy due to the erratic motion of players and their body parts. Figure 3.5 gives the ROC curve of both constraint classifiers. For high precision rates of the ROC curve, the constraint classifier gives much higher recall rates when using the spatial layout of supervoxels than when using optical flow.

**Sensitivity Analysis.** Table 3.6 evaluates sensitivity to the only input parameter of CSL—
Table 3.2: VolleyBall dataset. Results of the proposed learning algorithm using both unnormalized and normalized features.

<table>
<thead>
<tr>
<th>Method</th>
<th>MD</th>
<th>FP</th>
<th>ID-sw.</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSL-VOX</td>
<td>2.69</td>
<td>2.49</td>
<td>0.41</td>
<td>94.41</td>
</tr>
<tr>
<td>CSL-Uniform</td>
<td>2.99</td>
<td>2.89</td>
<td>0.45</td>
<td>93.67</td>
</tr>
<tr>
<td>CSL-UnNormal</td>
<td>10.87</td>
<td>13.64</td>
<td>17.28</td>
<td>58.21</td>
</tr>
<tr>
<td>CSL-UnNormalTrain</td>
<td>2.73</td>
<td>2.61</td>
<td>0.59</td>
<td>94.08</td>
</tr>
</tbody>
</table>

Table 3.3: VolleyBall dataset. Results of different variants of CSL.

<table>
<thead>
<tr>
<th>Method</th>
<th>MD</th>
<th>FP</th>
<th>ID-sw.</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>37.58</td>
<td>36.97</td>
<td>6.06</td>
<td>19.39</td>
</tr>
<tr>
<td>CSL-noProp</td>
<td>20.91</td>
<td>20.60</td>
<td>16.06</td>
<td>42.43</td>
</tr>
<tr>
<td>CSL-ConsFeat</td>
<td>20.79</td>
<td>20.43</td>
<td>17.41</td>
<td>41.37</td>
</tr>
<tr>
<td>CSL-noColor</td>
<td>3.04</td>
<td>2.89</td>
<td>2.76</td>
<td>91.31</td>
</tr>
<tr>
<td>CSL-noFlow</td>
<td>7.26</td>
<td>8.61</td>
<td>22.54</td>
<td>61.59</td>
</tr>
</tbody>
</table>

the number of mid-level features (e.g. supervoxels in this case)—on the Volleyball dataset. As expected, increasing supervoxels improves accuracy and increases runtimes.

**Runtime.** The average runtime of generating supervoxels and dense point trajectories is about 28s/frame and 24s/frame respectively. Given mid-level features, Table 3.6 shows that the runtime for CSL is quite fast. Further, CSL is approximately 10x faster when using constraint propagation compared to no propagation. This is because constraint propagation sets the label of approximately 9/10 of the supervoxels, which is a 90% reduction in the number of cost function evaluations. This agrees with the intuition that most tracking decisions are quite easy, while only in certain cases, such as occlusion, do we need to carefully consider data association.

**Qualitative evaluation:** Figure 3.7(a) illustrates tracking results for CSL and the baselines on a Volleyball video where the orange player is moving left, the yellow player is moving right, and the purple player stays still. Only our method gives correct tracks in this case. NCuts is sensitive to the choice of the number of clusters. Also, NCuts may produce solutions that violate the hard constraints, since they are “softened” in the affinity matrix. Network flow* and Superpixels confused the ID’s of the players on the two sides.

A failure example of our approach in the Volleyball dataset is shown and described in Fig-
Table 3.4: Results for the Figment dataset. PRCE: percentage of wrongly labeled pixels per player mask; Recall: percentage of recalled pixels per player mask; Tracking time: the number of frames where recall is above 20%.

<table>
<thead>
<tr>
<th>Method</th>
<th>PRCE</th>
<th>Recall</th>
<th>Tracking time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSL-VOX</td>
<td>17.10</td>
<td>82.89</td>
<td>89.73</td>
</tr>
<tr>
<td>CSL-DPT</td>
<td>19.28</td>
<td>43.62</td>
<td>79.41</td>
</tr>
<tr>
<td>[36]</td>
<td>20.32</td>
<td>31.07</td>
<td>75.13</td>
</tr>
<tr>
<td>[16]</td>
<td>86.42</td>
<td>0.46</td>
<td>1.03</td>
</tr>
</tbody>
</table>

Table 3.5: Results for PETS2009 S2L1

<table>
<thead>
<tr>
<th>Method</th>
<th>Rec.</th>
<th>Prec.</th>
<th>ID-sw.</th>
<th>MOTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSL-VOX</td>
<td>98.28</td>
<td>91.07</td>
<td>6</td>
<td>89.78</td>
</tr>
<tr>
<td>CSL-DPT</td>
<td>97.64</td>
<td>90.45</td>
<td>8</td>
<td>88.13</td>
</tr>
<tr>
<td>[12]</td>
<td>60.00</td>
<td>81.00</td>
<td>28</td>
<td>80.00</td>
</tr>
<tr>
<td>[89]</td>
<td>90.81</td>
<td>90.66</td>
<td>19</td>
<td>81.46</td>
</tr>
<tr>
<td>[7]</td>
<td>85.13</td>
<td>96.28</td>
<td>15</td>
<td>81.84</td>
</tr>
<tr>
<td>[42]</td>
<td>98.6</td>
<td>98.5</td>
<td>10</td>
<td>96.60</td>
</tr>
<tr>
<td>[107]</td>
<td>96.45</td>
<td>93.64</td>
<td>8</td>
<td>90.3</td>
</tr>
</tbody>
</table>

3.6 CONCLUSION

We presented a new approach for multi-object tracking under significant occlusion based on labeling mid-level, temporal features such as supervoxels. This labeling problem relaxes common assumptions of existing data-association methods. The main assumption of our approach is that the features provide an over-segmentation of the video. One of our main contributions is the constrained sequential labeling (CSL) approach, which leverages hard constraints and a flexible cost function to perform accurate labeling. We provided learning algorithms for constraints and cost functions and proved that cost-function learning converges to an accurate solution in finite-time when such a solution exists. Our experimental results in volleyball, basketball, and pedestrian tracking demonstrate that the approach is superior to the state-of-the-art when people detectors are unreliable and comparable to the state-of-the-art even when detections are accurate.
Figure 3.5: ROC curve for the constraint classifier when either using the spatial layout of supervoxels, or raw optical flow as input features. In particular, for precision of 97.3%, the former recall is 44.57%, and the latter recall is 8.04%. For generating responses of our constraint classifier, we operate in the region of high precision rates.

<table>
<thead>
<tr>
<th>Number of supervoxels</th>
<th>accuracy</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>-96.56</td>
<td>23s</td>
</tr>
<tr>
<td>1000</td>
<td>-91.20</td>
<td>76s</td>
</tr>
<tr>
<td>1500</td>
<td>84.66</td>
<td>124s</td>
</tr>
<tr>
<td>2000</td>
<td>92.28</td>
<td>291s</td>
</tr>
</tbody>
</table>

Table 3.6: Results of tracking accuracy and running time of our approach with different number of supervoxels as input for videos with 300 frames.
Figure 3.6: One failure example. Here the orange player occludes the yellow player, but our approach labels most of the orange player yellow. One reason is that our background subtraction removes the supervoxel corresponding to the player’s leg from our list of foreground supervoxels. At the same time, the two players are both moving left, so none of our features used for computing affinities between supervoxels provides useful information.
Figure 3.7: Volleyball dataset: first row(CSL-VOX), second row(Ncut), third row(Superpixel), fourth row(Network Flow*)
Figure 3.8: Our tracking results in the Figment dataset
Chapter 4: Under-Segmentation to Tracking

4.1 INTRODUCTION

While the aforementioned CSL performs well when a reasonable over-segmentation exists, for videos where the targets are of relatively low resolution, it is usually very hard to obtain such an over-segmentation. So in this work, we target at these videos. Generally tracking multiple targets in such low resolution videos are very hard, so in this work, we step one back and study the general problem of detection. Instead of using an over-segmentation of the video, we explore how under-segmentation can be used. Results from an under-segmentation of the video provides segments where each segment may contain one or multiple targets. One major problem with under-segmentation is that it is not clear how many objects are covered in each segment. This motivates us to conduct this work.

In this work, we introduce a new problem, person count localization from noisy under-segmentation of the video (e.g. foreground masks) and person detections. Our formulation can be viewed as a middle-ground between person detection and frame-level counting. Given a video, our goal is to output for each frame a set of:

1. Detections optimally covering both isolated individuals and crowds of people in the video; and

2. Counts assigned to each detection indicating the number of people inside.

Person detection is an important line of research, since detecting people in video frames has become the standard initial step of many approaches to activity recognition [3, 59, 37, 5, 24, 9, 92], and multi-object tracking by detection [73, 104, 90, 46, 6, 102, 42]. They typically use as input human appearance, pose, and orientation, and thus critically depend on robust person detections. In many domains, however, such as videos of American football (Fig.4.1) or public spaces crowded with pedestrians, detecting every individual person is highly unreliable, and remains an open problem.

This motivates us to study this alternative formulation that do not require perfect person localization, especially under severe clutter and occlusion, and still prove useful for higher-level
Figure 4.1: Our count localization results for an image sequence from American football. Challenges include severe occlusion, clutter, and similar appearance of players.

video understanding like tracking. One related problem that has successfully addressed videos of crowded scenes is frame-level counting of people [10, 85, 61, 20, 56]. This frame-level count information, however, is a very coarse description of the video, with limited utility for high-level tasks. In particular, these problem formulations and evaluations do not address location of individuals or sub-groups.

Rather than counting people per frame, we would like to retain as much localization capability of detecting individuals as possible, but gracefully transition to counting people within areas in the frame occupied by crowds. As these crowded groups are often isolated and visually distinct from the rest of the scene, they can be viewed as “visual phrases” whose spatially tight localization and count assignment could provide useful cues for higher-level processing. For example, as shown in Fig.4.1, localized counts provide rich information about the activity unfolding during a football play by identifying many isolated and small groups of players and the primary larger player groups. Similarly, localized counts can provide space-time density statistics of crowds in an area of interest and also serve as a basis for more refined individual tracking when desired.

**Overview of Approach and Contributions.** Our approach first extracts noisy foreground by running a person detector and foreground segmentation, which will sometimes be redundant. A flow graph is then built and transformed into a integer linear program (ILP). The construction of this ILP is our first contribution as it must deal with the redundancy and false positives in input.

Our second contribution is to improve the initial solution via the new framework of iterative error-driven graph revision (EGR). The key idea is that the ILP is derived from a flow graph whose structure is based on hard-to-tune parameters and noisy input. As a consequence, for any fixed graph construction approach, there will be cases where the ILP solution can be observed to have visually obvious errors that violate basic integrity constraints (e.g. a disappearing person
in the middle of a scene). To address this issue, EGR iteratively constructs a sequence of such graphs, and the corresponding integer programs, such that each new flow graph is a refinement of the previous graph, and aimed at correcting violated integrity constraints of the previous solution. We use a simple learning strategy to select among various refinements available at each step, such as running a tracker to produce new detections in an area or adding edges to the graph. The EGR process stops when no further integrity constraints are found or refinements are selected and in our experiments produce significantly improved results.

Our third contribution is to introduce a new metric for person count localization that accounts for both errors in localization and counting. We provide experiments in two challenging domains: American football and pedestrian crowds. The results demonstrate the benefits of our approach compared to prior work and a number of baselines. Also, our evaluation suggests that EGR is a promising framework for improving other vision tasks based on fixed compilations to optimization problems.

4.2 PRIOR WORK

While we are the first to define the problem of person count localization and associated evaluation metrics, prior work has studied related problems.

In multi-object tracking, counting problems are sometimes solved as part of the overall approach to deal with groups of people. [73] greedily assigns and propagates counts, which can lead to inconsistent counts. Instead, [42] formulate counting as a flow problem but uses a simple linear cost function based on detection size, which is not appropriate when there is heavy occlusion or the number of counts vary significantly. Similar to [42], we also formulate our count localization problem in a flow framework but we have a more complex objective function and constraints and place an additional focus on being robust to segmentation/detection noise. The most similar work to ours was on the problem of biological cell tracking, where maintaining cell counts in foreground blobs is the main step followed by heuristic id maintenance [86]. They solved the problem by formulating an integer program based on a flow graph over foreground blobs. Their approach, however, is insufficient for our problems, as our experiments show. The key issue is that contrary to our primary motivating application of team sports, their cell tracking application allowed for high quality non-intersecting foreground blobs with few false positives and negatives, which significantly simplifies the problem. In our application of American football, foreground extraction is often quite noisy, producing high rates of both false positives and
negatives.

The crowd counting problem generally has the goal of accurately counting the number of people in a frame. Prior work [10, 85, 61, 20, 56, 54] typically follows a two-step pipeline where foreground segments are first extracted and then counts are independently estimated for each segment based on local features. Research has focused mainly on feature design and training reliable estimators, but has mostly ignored the consistency and interactions between segments. While some of these methods are able to generate counts for more localized segments, the evaluations have all focused on the accuracy of total counts, which ignores localization performance.

4.3 APPROACH

We now describe our approach to the person count localization problem, where the input is a video and the output is a set of detections, each labeled by a count of people. The quality measure of the output is based on both the accuracy of the counts assigned to detections and the localization of the detections. Intuitively, we desire count-one detections to be associated with individual people when possible, but for cluttered crowds we are satisfied with a crowd-level detection and accurate count. In our experiments, we introduce a new evaluation metric to capture these concerns.

4.3.1 OVERVIEW

Our iterative solution approach, error-driven graph revision (EGR), is depicted in Fig. 4.2. In the first iteration we extract foreground objects/blobs from the video and build a corresponding initial flow graph representation $G_0$ that represents the temporal-spatial relationships among the foreground objects. An integer linear program (ILP) is then formulated based on $G_0$ that both selects a subset of detections and assigns counts to them, giving a solution denoted by $C_0$. The ILP is designed with the goal of maintaining accurate counts that also maintain temporal-spatial consistency.

At iteration $i$ of EGR, we first look at the ILP solution $C_{i-1}$ from the previous iteration in order to identify violations of common-sense integrity and domain constraints (for example a person cannot appear or disappear in the middle of the frame). Such violations are inevitable in our experience for any fixed way of constructing graphs from the input. Associated with each type of constraint violation are potential graph-revisions operations that may address the viola-
Figure 4.2: System Overview. First extract noisy foreground by running an object detector and foreground segmentation. A flow graph is then built and transformed into an integer program. In the following iterations, the approach detects places where integrity/domain constraints are violated by the current solution and applies one or more graph-revision operators to obtain a new graph and updated solution.

Note that, we do not assume the initial extracted foreground objects to be perfect. In fact, the iterative process aims at dealing with this noisy input. As we will see later, the ILP can help address the problem of false positive foreground objects by not selecting them or assigning them counts of zero. However, the ILP does not have a natural way to deal with false negatives, which do not even appear in the corresponding flow graph. The key idea behind the EGR approach is that the ILP solutions in such cases will often violate common-sense integrity constraints that can be easily checked. Further, for a detected violation, there are natural ways to revise the graph that will potentially correct the violation, for example, by using a tracking mechanism to acquire detections in a certain space-time region of the video that were missed by the initial processing.

We note that an alternative to EGR would be to construct a single graph $G^*$ and ILP that accounts for all possible missing detections and edges. This, however, is impractical for at least two reasons. First, the enormous number of such possibilities would stress even state-of-the-art IP solvers. Second, the number of false-positives represented in the graph would grow dramatically, resulting in less reliable solutions due to the increase in ambiguity. Rather, EGR can be viewed as an approach that aims to incrementally construct a graph $G$ containing only the “necessary” parts of $G^*$ for a particular problem instance.
4.3.2 FOREGROUND DETECTION AND GRAPH BUILDING

Given a video, we first need to extract foreground detections that will serve as candidate detections to be labeled by counts. In contrast to previous work that either runs an object detector or performs foreground segmentation to obtain the foreground detections, we apply both a person detector and foreground segmentation. As shown in Fig.4.3, these two methods have their own strengths. A person detector usually works well for single isolated people, but has problems when there is occlusion or inside a crowd, while foreground segmentation usually works well when there is a clutter of people but performs poorly for smaller object due to noisy background modeling and registration. We combine the two methods by using both the person detector’s results and the relatively larger connected components from the foreground segmentation. In this way, we can get an initial set of foreground detections with a reasonable recall. Note that there will often be significant overlap between the foreground and detections, which is a complication that the optimization process must account for.

We denote all the foreground detections obtained from the two methods by \( \{d^t_i\} \) where \( t \) is the frame index. We then build a flow graph \( G = (V, E \cup E') \) as shown in Fig.4.2. Each foreground detection is represented by two vertices \( u^t_i \) and \( u^t_i' \). There are two types of edges in \( G \). The solid edge set \( E \) contains edges \( e_{i,i'}^{t,t} \) and \( e_{i,j}^{t,t+1} \) where \( e_{i,i'}^{t,t} \) links \( u^t_i \) to \( u^t_{i'} \). \( e_{i,j}^{t,t+1} \) link \( u^t_{i'} \) to a subset of \( u^t_{j+1} \) in the next frame. The linkage is determined by the following rules: we first link foreground detections that form reliable tracklets as suggested by [58]. For other foreground detections, \( e_{i,j}^{t,t+1} \) is added if \( d^t_j \) is in the neighborhood of \( d^t_i \) according to a threshold. Note that, a particular threshold may not work for all cases because of different viewpoints and perspectives. Our approach will adaptively change this threshold more locally in later EGR stage if needed. The other set of dashed edges \( E' \) are hyper edges and link a subset of \( e_{i,i'}^{t,t} \) in the same frame. Since our foreground detections can overlap, we add a hyper edge \( e_{i,j}^{t,t} \) between \( e_{i,i'}^{t,t} \) and \( e_{j,j'}^{t,t} \) if for \( d^t_i \) and \( d^t_j \), one is covered by the other or their intersection of union score is larger than a threshold.

4.3.3 INTEGER PROGRAMMING FORMULATION

We now wish to convert the flow graph to an optimization problem that when solved will assign counts to the detections represented in the graph in a way that maximizes the estimated count accuracy while satisfying basic flow constraints. Given the above graph \( G \), consider assigning
each edge in $E$ a corresponding variable $x$ indicating the amount of flow (number of people) going through that edge, for example, $x_{t,i,i'}^{t,t}$ is a variable indicating the flow across edge $e_{t,i,i'}^{t,t}$. Note that the flow assigned to $x_{t,i,i'}^{t,t}$ is interpreted as the count of people assigned to detection $d_i^t$.

Given these variables we would like to find flow values (equivalently count values) that result in consistent flows and also maximize some measure of count accuracy for each detection.

In order to measure count accuracy we use a function $f_{t,i,i'}^{t,t}(x_{t,i,i'}^{t,t})$ that assigns an accuracy score to the count assigned to $d_i^t$. In traditional network flow formulations, these functions are linear in $x_{t,i,i'}^{t,t}$ and in that case yields a polynomial time algorithm. However, for our problem it is unlikely that any linear function will approximate $f$ well, since the accuracy of a count assignment is going to non-trivially depend on the visual evidence associated with detection $d_i^t$.

In this work, we define $f$ values based on the confidence of learned random forest classifiers. In particular, given an upper bound $N$ on the maximum count that can be assigned to a detection, we train a random forest based on labeled training data to predict the discrete count $\{0, \ldots, N\}$ for a detection $d_i^t$ given visual features of that detection. The value of $f_{t,i,i'}^{t,t}(x_{t,i,i'}^{t,t})$ is then taken to be the confidence that $d_i^t$ should be assigned count $x_{t,i,i'}^{t,t}$. The prediction of the random forest is based on the following features: detection type (whether the detection is from a person detector or foreground), location, size, number of foreground pixels, and the spatial distribution of foreground pixels. We note that more sophisticated regression algorithms that provide confidences could also be used to define $f$.

Given this definition of $f$ we would like to find the flow assignment $x$ that maximizes the total estimated count accuracy $\sum_{i,t} f_{t,i,i'}^{t,t}(x_{t,i,i'}^{t,t})$ subject to standard flow conservation constraints that make sure the counts are consistent across frames. Further, we also want to enforce con-
straints corresponding to the hyper edges in $G$, which state that we only want to assign non-zero counts to one detection in a pair of overlapping detections. Unfortunately, in contrast to standard network flow formulations where $f$ is linear, when $f$ is relatively arbitrary as in our case, the problem of optimizing the count accuracy objective is NP-complete. This means that we are unlikely to find an efficient exact algorithm. However, below we formulate this problem in terms of an integer linear program (ILP), which allows us to apply state-of-the-art ILP solvers to our problem.

To formulate our problem as an ILP we introduce indicators for each flow variable to linearize the objective. We denote $x_{t,i,i',n}$ to be the indicator of flow variable $x_{t,i,i'}$ taking value $n$ and similarly for $x_{t,i,j,n}$. We also define $e_{t,i,i',n}$ to be the accuracy score of $f$ for assigning detection $d_i^t$ a count of $n$. The ILP can now be defined as follows:

$$\begin{align*}
\max \quad & \sum_{i,t} \sum_{n} c_{t,i,i',n} x_{t,i,i',n} \\
\text{s.t.} \quad & \text{for all } i, t \\
& \sum_{n} x_{t,i,i',n} \leq 1, \sum_{n} x_{t,i,j,n} \leq 1 \text{ for } e_{t,i,j,n} \in E, \quad (a) \\
& \sum_{n} n x_{t,i,i',n} = \sum_{j: e_{t,i,j+1} \in E} \sum_{n} n x_{t,i,j,n}, \quad (b) \\
& \sum_{n} n x_{t,i,i',n} = \sum_{j: e_{t,i,j-1} \in E} \sum_{n} n x_{t,i,j,n}, \quad (c) \\
& \sum_{n} x_{t,i,i',n} + \sum_{n} x_{t,j,j',n} \leq 1 \text{ for } e_{t,i,j} \in E', \quad (d) \\
& 0 \leq x, x \in \mathbb{I} \quad (e)
\end{align*}$$

The set of constraints (a) make sure all indicators for one edge sum up to less or equal to 1. The flow conservation constraints correspond to (b) and (c) and the hyper edge constraints correspond to (d).

With the hyper edge constraints, not all input detections will be assigned a count in the ILP solution. When there is a large group with a large detection from segmentation and a few smaller, overlapping detections from the person detector, the large detection will usually be chosen as it better facilitates the flow constraints. However, when foreground extraction is noisy, sometimes these large detections can contain significant areas that do not contain people and also people
that can be localized by people detectors. In such cases, we could get improved localization by also using the overlapping smaller detections. To account for this, after solving the ILP, we perform an additional optimization at the detection level for any detection \( d \) that contains smaller detections. In particular, given such a detection \( d \) with a count value of \( c \) from the ILP solution, we wish to best assign counts to the smaller detections in order to provide better localization within \( d \). We use a greedy optimization for this and greedily assign counts to the small detections in order to maximize their \( f \) scores with the constraint that the total of the counts does not exceed \( c \). After doing this, if the total count \( c' \) assigned to the smaller detections is less than \( c \) we assign \( d \) a count of \( c - c' \) indicating that the remaining people are somewhere in \( d \) but not precisely localized.

4.3.4 ERROR-DRIVEN GRAPH REVISION

The initial set of detections we get are noisy and there can be both false positives and missing detections. The ILP attempts to address the problem of false positives by allowing for counts of zero to be assigned to any detection. However, the ILP has no way of dealing with missing detections. In addition, as we mentioned above, the ILP relies on the graph \( G \) which is built based on certain thresholds, which are hard to define so as to work well in all situations. It is thus, desirable to be able to adjust the thresholds locally if the need is detected. In order to deal with these problems, we introduce the iterative EGR framework.

To apply EGR one must specify integrity constraints that hold for (nearly) all solutions. The constraints can come from common sense or domain-specific knowledge. In our case, we use the simple constraint that people cannot appear/disappear at non entry/exit locations. In our current domains, this single constraint was sufficient to allow EGR to significantly improve performance. Note that such domain constraints are often hard to directly impose in the ILP while retaining non-trivial solutions due to missing detections. However, they are easy to check given an ILP solution. In our case, we simply look for foreground detections that have non-zero counts assigned and have no successor in the next frame or predecessor in the previous frame in the graph. We denote these detections by \( \{d_{ei}\} \).

Next, we need to update these places. As we mentioned, there are two error sources, one is missing detections and the other is inappropriate thresholds used in graph construction. We propose three operators to correct these errors.

**Add a node** (Fig.4.4 top). This operator applies when we have a small gap of missing
detection. If we decide to apply this operator at $d_{e_i}$, we will create a new detection that is identical to $d_{e_i}$ in the next (previous) frame, and then modify the location according to a constant velocity model.

**Add a tracker** (Fig.4.4 middle). When we are missing foreground detections for multiple frames, we fire an object tracker at $d_{e_i}$ to track the target forward (backward). We stop tracking when the tracker is not confident or the tracking result overlaps with existing foreground detections. We then add all the tracking results to the graph. The idea is that the tracker behaves as a localized detector for $d_{e_i}$ that can overcome mistakes made by the more general detectors.

**Add an edge** (Fig.4.4 bottom). When there is foreground detections around $d_{e_i}$ in the next (previous) frame, we might just lower the threshold of the graph construction and add an edge between $d_{e_i}$ and some existing foreground detections.

It is not straightforward to decide which operator to apply, especially for adding a node and adding an edge. So we train a random forest classifier to mimic the choices made by an experienced human in various situations. The features for the classifier include distance to the existing closest detection in the next (previous) 1 frame and 5 frames, size of uncovered foreground pixels in the neighborhood in the next (previous) 1 frame and 5 frames, and sum of optical flow magnitude within the detection. Training examples were generated by creating ILP solutions on a training set, finding integrity constraint violations and then have the human label them by the most appropriate operator.

After making these local updates to the graph we create a new ILP and rerun the solver initialized with the previous solution. We iterate over solving the ILP and updating the graph until there is no error detected in the solution or a certain number of iterations is reached.

### 4.4 EXPERIMENTAL RESULTS

**Datasets.** We evaluate on two datasets from the domain of American football and a pedestrian domain. The football dataset contains 10 videos from a football game where each video depicts a complete play ranging from 200 to 400 frames with resolution $852 \times 480$. The challenges here include large view-point variations, fast camera motions and complex player interactions. Ground truth bounding boxes are labeled for all 22 players and 1 defensive referee. The pedestrian dataset is taken from [10] and depicts pedestrians walking in two directions along a sometimes crowded walkway. The camera is stationary and the video contains 2000 frames with a resolution of $238 \times 158$. The ground truth location for each pedestrian is also provided for eval-
uation. The challenge here is that the density of people is high and there are seldom isolated people. Compared to the football domain, however, the foreground segments provided are much less noisy.

**Implementation Details.** For football, the foreground segmentation is done by automatically registering each frame to a panorama of the football field and then doing background subtraction to obtain foreground blobs. We also used training videos from the same game to train a DPM detector [34] to recognize players. The count prediction model required to produce scores for our ILP is obtained by training a random forest classifier on 4 videos with different viewpoints and then testing on the remaining videos. The precision for the initial input foreground detections is 83.36%. 1.2 players out of 23 are not covered by any initial foreground detection in each frame on average. For the pedestrian dataset [10], we use the same foreground segmentation from [10] and train a Haar detector to detect individual people. We follow the same training strategy as [10] and use frames from 600 to 1399 to train the random forest classifier. The single object tracker used in our EGR framework for both datasets is from [109]. We use the Gurobi ILP solver and perform a maximum of 5 EGR iterations.

**Baselines.** For the football dataset, we compare with several variants of our approach: 1) \(EGR\), our full count localization approach, 2) \(RF\), we use the trained random forest to assign count to every input detection independently without enforcing flow constraints, 3) \(EGR_i\), our approach run for \(i\) iterations, in particular, \(EGR_1\) is the results of our ILP with the initial input, 4) \(EGR_{i-,e-,t-}\), our approach without applying one of the operators (adding nodes, edges, trackers) respectively, 5) \(EGR^*\), our approach applied to the ground truth foreground, i.e. each input detection is a connected component in the ground truth foreground, 6) [86], we implement the method described in [86] and made it work for the football domain. 7) [86]*, [86] applied to the ground truth foreground. For the pedestrian dataset [10], we compare against prior state-of-the-art results.

**4.4.1 EVALUATION METRICS**

There are no existing metrics designed to measure the performance of count localization. Prior work such as [86] did not measure the count performance explicitly but rather the overall event-recognition system. Crowd counting work such as [10, 66] focus on global count accuracy without regard for localization. Thus we propose a new metric called **count localization accuracy (CLA)** that is aimed to evaluate both count and localization accuracy.
Suppose for one frame, we have $n$ ground truth people and produce a solution with $m$ detections each with a count $c_i$. To calculate the metric, we first greedily match ground truths to detection results. If the intersection over union score (IoU) between a ground truth and a detection is over a threshold, we call this a match candidate, and each ground truth is matched to only one detection with the highest IoU score. A detection with count $c$ cannot be matched to more than $c$ ground truths. After the matching is done, for each ground truth, we calculate the IoU score $s_i$ between the ground truth and its corresponding detection (0 if there is no match). The metric is then calculated as: 

$$CLA = \frac{\sum_{0 \leq i} s_i}{\sum_{0 \leq i} c_i + n'},$$

where $n'$ is the number of unmatched ground truths. In the ideal case, where our results are the same $n$ detections as ground truth and each with count 1, the metric gives a score of one. This metric evaluates both counting precision and localization. In one extreme case where we have all the correct detections but wrong counts, the denominator gives a penalty. In another extreme case where we have a single large detection that covers all ground truths with the correct count, the numerator which evaluates the IoU gives a penalty.

In addition to CLA we also report some other common metrics. Localization Accuracy (LA) keeps the same numerator as CLA and uses ground truth counts as the denominator and thus only accounts for localization accuracy. Missing count (MC) calculates the percentage of the ground truth that are miss counted. Count error (CE) calculate the average absolute count error. For the football dataset, we compute CE based on each output detection and denote it as $CE_{d}$. For the pedestrian dataset, CE is computed based on the entire frame to be able to compare with previous results and is denoted as $CE_{f}$.

### 4.4.2 RESULTS

**Quantitative Results:** Tab.4.1 shows results for the football dataset. First, we observe that our approach EGR outperform [86] since [86] has no way to deal with missing detections in the noisy input. In fact, [86] performs similar to our approach in the first pass, i.e. without any graph revision and working with initial input. Further, when applying to the ground truth foreground, [86] has the same performance as ours. It shows that under ideal input, even if [86] has a more complex objective (transition score), we are able to achieve the same results. As a reference point we also give results for an oracle frame-level approach *Frame* that has a single detection over all people with the correct count. We see EGR achieves a CLA much closer to the
Table 4.1: Results for American Football dataset. Maximum number of targets is 23 per frame.

<table>
<thead>
<tr>
<th>Method</th>
<th>CLA</th>
<th>LA</th>
<th>MC</th>
<th>CEd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame</td>
<td>0.0112</td>
<td>0.0112</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EGR*</td>
<td>0.1718</td>
<td>0.1976</td>
<td>0</td>
<td>0.15</td>
</tr>
<tr>
<td>EGR</td>
<td>0.1551</td>
<td>0.1830</td>
<td>0.04</td>
<td>0.18</td>
</tr>
<tr>
<td>RF</td>
<td>0.1101</td>
<td>0.1562</td>
<td>0.22</td>
<td>0.49</td>
</tr>
<tr>
<td>EGR1</td>
<td>0.1166</td>
<td>0.1506</td>
<td>0.24</td>
<td>0.34</td>
</tr>
<tr>
<td>EGR2</td>
<td>0.1387</td>
<td>0.1811</td>
<td>0.13</td>
<td>0.38</td>
</tr>
<tr>
<td>EGRn−</td>
<td>0.1273</td>
<td>0.1668</td>
<td>0.06</td>
<td>0.31</td>
</tr>
<tr>
<td>EGRt−</td>
<td>0.1186</td>
<td>0.1684</td>
<td>0.20</td>
<td>0.42</td>
</tr>
<tr>
<td>EGRc−</td>
<td>0.1471</td>
<td>0.1753</td>
<td>0.05</td>
<td>0.34</td>
</tr>
<tr>
<td>[86]</td>
<td>0.1205</td>
<td>0.1562</td>
<td>0.23</td>
<td>0.25</td>
</tr>
<tr>
<td>[86]*</td>
<td>0.1718</td>
<td>0.1976</td>
<td>0</td>
<td>0.15</td>
</tr>
</tbody>
</table>

To provide a better understanding of our CLA metric. Frame is an oracle frame-level approach. oracle EGR* than to Frame. When applying the random forest classifier RF independently to detection, the count error increases. Comparing different iterations of EGR, we can see that we have a big performance boost from the first pass to second pass. This is mainly because with the tracker, we recover many missing detections. Among the three operators we have, the tracker plays the most important role under a limited number of iterations. It is interesting to note that for the football dataset over 70% of the detections output by our approach EGR have an assigned count of 1 and over 90% have a count of 4 or less. This shows that the detections output by the approach are generally quite localized to individuals or small groups. Detailed results can be found at Fig. 4.5.

Tab. 4.2 shows results for the pedestrian dataset. In terms of frame-level counting error for left-traveling and right-traveling people, which was the focus of prior work, our frame-level performance is a bit worse than two of the prior systems [10, 66]. In this dataset, the foreground segmentation provided by [10] is quite accurate compared to the football dataset, which means that our graph revision framework plays a much less important role here. Instead, the performance of the local random forest classifier becomes the dominating factor. Since our random forest classifier is much simpler than the regressors developed for previous work [10, 66], which was their primary focus but not ours, this result is not surprising. However, our approach is able to provide localization information in addition to frame-level counts. Since we cannot compute CLA scores for these frame-level approaches we report in the table the CLA for the oracle frame-level approach that provides 2 detections each covering all foreground segments in one direction.
Table 4.2: Results for Pedestrian dataset. * We calculated CLA for [10, 26, 66] assuming they output 2 detections, one for each direction, and each direction has the correct counts.

and are assigned the correct count. We see that the CLA score of our approach is orders of magnitude larger. Our localization performance is also depicted in Fig.4.6, which illustrates the heat maps of counts for each pixel for the pedestrian dataset. We can draw some somewhat obvious but interesting conclusions from these maps. For example, people in large groups tend to walk in the middle of the walkway. Also, most people walk along their right hand side of the walkway.

**Runtime.** The run time of our approach varies from one to 20 minutes for different videos on a desktop with a 4-core 3.4GHZ CPU. The majority of time is spent on the ILP solver. This suggests that the best way to speedup our current system would be to investigate approximate and fast solution techniques for the ILP problems we generate.

**Qualitative Evaluation:** Fig.4.7 illustrates count localization results on an example American Football video. [86] has similar results as $EGR_1$ and misses several players due to the noisy input. Some of the under-counting we see here is also due to missing players in previous frames. Our approach is able to identify these errors in the input and correct them through EGR. Similarly, Fig.4.8 shows our results on some frames of the pedestrian dataset. With the help of the person detector, we are able to get more localized counts.

### 4.5 CONCLUSION

We formulated the new problem of person count localization that is useful for crowded domains with severe occlusion and interference among people such as team sports domains. We presented an approach to this problem called iterative error-driven graph revision, which attempts to overcome noisy input detections by detecting integrity constraint violations and then adjusting the optimization problem appropriately. This idea was shown to be useful in our experiments and more generally may be useful in other applications where global optimization problems are formulated from vision data. We introduced a new metric called count localization accuracy for evaluating the localization and count quality of solutions and evaluated our approach on datasets.
derived from American football and moving pedestrians. The results show that our approach is significantly better than competitors when the input is noisy and that for the much less noisy pedestrian dataset our approach was competitive in terms of frame-level counts while also providing localization information.
Figure 4.4: Illustration of three operators. Top: Add a node. Miss a detection in one frame while there are corresponding detections in neighboring frames. Middle: Add a tracker. A target continue missing for several frames. Bottom: Add an edge. This should be a merge, but we did not connect initially because of inappropriate threshold.
Figure 4.5: Histogram of counts for football (left) and UCSD (right) dataset.

Figure 4.6: Heat map of counts for the pedestrian dataset. Left: left direction, Right: right direction. See Fig.4.8 for image of scene.
Figure 4.7: An example image sequence from American football dataset. First row: initial input of foreground detections, second row: ([86]), third row : \textit{(EGR)}_1, fourth row \textit{(EGR)}. [86] and \textit{EGR}_1 miss a few players because of the initial input. \textit{EGR} is able to recover these miss detections by iteratively revise the graph.
Figure 4.8: Pedestrian dataset. Green and Red boundaries outline the foreground blobs moving left and right. Green bounding boxes show smaller detection from greedy assignment.
Chapter 5: Deep Tracking-by-Segmentation

5.1 INTRODUCTION

The success of deep networks has inspired recent work on both tracking via bounded boxes [97, 65, 94, 96] and segmentation [41, 74, 78]. However, there are still no deep networks that directly solve the tracking-by-segmentation problem. The main contribution of this work is to propose the first deep architecture for online tracking-by-segmentation. Unlike the existing deep segmentation networks, it does not require that the target belongs to a known object class for which the network is trained on. Unlike deep tracking networks, instead of outputting a bounding box, our network predicts a figure-ground mask that classifies each pixel in the frame. Our network consists of a composition of a CNN and RNN. Given the predicted mask in the previous frame, the CNN first predicts a coarse segmentation mask for the current frame. This prediction is then refined by the RNN with the help of an addition edge image. Note that a fundamental difference between our CNN and most previous work is that the input to the network contains the previous prediction, which complicates training due to the dependence between the predictions. Another challenge in training such a network is that most existing video segmentation datasets only have partial annotations, where only a fraction of the video frames are labeled. In order to address these challenges, we use an iterative training algorithm that aggregates data from multiple training runs and also propose a data augmentation algorithm that creates noisy ground truth masks for frames without annotation.

To summarize, there are two main contributions of this work. First, we propose a deep architecture for online tracking-by-segmentation. To the best of our knowledge, this is the first deep architecture for this problem. Second, we propose an iterative approach to generating training data for the deep architecture where the training instances are dependent and sparsely labeled. Our experiments on two challenging sets of benchmark videos show that our approach achieves higher or equal performance to the state-of-the-art and natural baselines.
5.2 RELATED WORK

End-to-end deep architectures have become a dominant technique in computer vision in recent years and deep networks have already been applied to both object tracking (via bounding boxes) and semantic segmentation. We have reviewed some recent deep tracking networks in the previous chapter and thus here we only focus on deep segmentation. Recent work [41] presents a hypercolumn representation for pixels that uses the corresponding activations from all layers of a deep network as a hypercolumn feature vector for classifying each pixel. A deconvolution network has been used for segmentation [74] where a series of unpooling and deconvolution layers are applied after a VGG net in order to recover the resolution and predict pixel-wise class labels. Another approach [78] considers a U-structure network that after the usual bottom-up pass conducts a top-down pass to refine the segment mask. Importantly, these segmentation networks are typically trained with respect to a certain set of object classes and thus cannot be used to segment objects outside of those classes, which limits their general applicability. Very recently, [50] trains a network to segment generic object in images. The network is adapted from an image classification network where the final classification layer is removed and the last two pooling layers are replaced with dilated conv layers which makes the network output a 8x reduction in final resolution. Bilinear interpolation is used to recover the resolution. The network is first trained on imagenet classification and then further trained on a semantic segmentation dataset where the ground truth masks with classes are transferred into binary masks of object vs background.

5.3 PROBLEM STATEMENT AND OVERVIEW

The input to our problem is a video with a segmentation mask for an arbitrary target object in the first frame. The goal is to track and segment the target object throughout the remainder of the video. We consider an online tracking-by-segmentation approach, where our deep network processes the video in a single pass from start to finish by outputting a predicted target mask for each frame. The network architecture consists of two subnetworks, a CNN for producing an initial prediction of the mask, followed by an RNN for refining the mask. Specifically, the first CNN subnetwork takes as input the current frame, the previous frame, and the predicted target mask for the previous frame. The output is a prediction of the target mask for the current frame. After the CNN predicts a target mask for the current frame, the second RNN subnetwork
iteratively refines the predicted target mask in order to improve the predicted target boundaries. Details of the network are presented in Section 5.4.1.

Since our network depends on its prior predictions when making a current prediction, several complications arise for training. First, the labeled training videos have ground truth target masks, but do not have the network predictions for previous frames, which are used by the CNN as input. Thus, it is unclear what values should be used for the CNN input during training. Second, the training samples for each prediction are not independent, contrary to typically assumptions in deep supervised learning. To solve these problems, we follow an approach similar to a recent algorithm for imitation learning. The Dataset Aggregation (DAGGER) algorithm [84] aims to address these issues by learning a sequence of deep networks. Each network is learned using training instances that are constructed using predictions from previously learned networks. More details of this training approach are in Section 5.4.2 along with our approach for handling sparsely labeled training data.

5.4 DEEP LEARNING FOR TRACKING-BY-SEGMENTATION

In this section, we first describe our deep architecture for tracking by segmentation. Next, we describe our approach for training that architecture from arbitrary training videos.

5.4.1 DEEP NETWORK ARCHITECTURE

Figure 5.1 illustrates the layered architecture of our CNN sub-network, which is adapted from the architecture used for a recent saliency network [76]. The input to the network stacks the current frame (3 channels), previous frame (3 channels) and previous target mask (1 channel) together. An important modification compared to the saliency network is that we have removed all pooling layers, which means that our network operates on fixed sized feature maps and there is no need to perform upsampling at the end to recover the image resolution. We have empirically found that removing the pooling layers is important for being able to recover finer details of object boundaries for our problem. Recent work has also noticed the benefits of removing pooling layers [63].

Our RNN architecture, which follows the CNN is shown in Figure 5.2, which is motivated by the RNN architecture used in prior work for iteratively improving classification maps [67]. The learned CNN produces a rough segmentation, but often one with inaccurate boundaries. We
use the RNN to iteratively refine the CNN output by having the RNN take into account a pre-computed edge image (produced by [29]). In each RNN iteration, the input to the network is an edge image and the predicted mask from last iteration, which is taken to be the CNN output in the first iteration. The network is trained to predict a modification to the mask, which is then added together with the input mask and fed forward to the next iteration. In our experiments, we use five refinement iterations of the RNN.

5.4.2 TRAINING

We follow a staged training process. First the CNN is trained to generate initial target masks. Next, the RNN is trained using the CNN target masks as the starting point for refinement. Finally, we train the combined CNN and RNN architecture jointly for fine tuning the parameters. Below, we first describe the loss function that we use for training. Next, we describe our training data generation method, which accounts for both the dependence between training data and for unlabeled video frames.

5.4.2.1 LOSS

Instead of using a softmax loss on the pixels, we choose to optimize our network over the segmentation-specific loss function, expected union over intersection (UOI). While this loss cannot be decomposed into pixel loss, it is still straightforward to derive its gradient with respect to each pixel. Next, we first define the loss in terms of pixel probabilities and then derive the gradient. Let \( \hat{p}_1^i \) and \( \hat{p}_0^i \) denote the predicted probability of the softmax layer of the \( i \)th pixel in the minibatch belonging to the target and background respectively and let \( p_1^i \) denotes the probability of the ground truth label for pixel \( i \) being foreground. If we denote \( l_i \) and \( \hat{l}_i \) to be the ground truth and predicted label for pixel \( i \), then the UOI loss can then be defined as:

\[
L = \frac{\mathbb{E}(\text{Union})}{\mathbb{E}(\text{Intersection})} = \frac{\mathbb{E}(\sum_i I(l_i = 1 \vee \hat{l}_i = 1))}{\mathbb{E}(\sum_i I(l_i = 1 \land \hat{l}_i = 1))} = \frac{\sum_i P(l_i = 1 \vee \hat{l}_i = 1)}{\sum_i P(l_i = 1 \land \hat{l}_i = 1)}
\]
\[ L = \frac{\sum_i (p^1_i + \hat{p}^1_i - p^1_i \hat{p}^1_i)}{\sum_i p^1_i \hat{p}^1_i} \]

where \( I() \) is an indicator function, \( E \) computes the expectation and \( p^1_i = l_i \). To simplify the equation, define \( S = \sum_i (p^1_i + \hat{p}^1_i) \) and \( M = \sum_i p^1_i \hat{p}^1_i \), which gives us the compact expression \( L = \frac{2S - M}{M} \). The gradient of the UOI loss with respect to the input probabilities can then be derived as:

\[
\frac{\partial L}{\partial \hat{p}^1_i} = \frac{(1 - p^1_i)M - p^1_i(S - M)}{M^2} = \frac{M - p^1_iS}{M^2}
\]

The gradients with respect to \( \hat{p}^0_i \) follows similarly.

While it is more common to use intersection over union (IOU) as the evaluation metric for segmentation, we have found that directly minimizing the IOU loss is less preferable compared to UOI. In particular, the gradient of IOU loss is defined as \( \frac{\hat{p}_i^1S - M}{(S - M)^2} \), which increases as \( \hat{p}_i \) approaches \( p_i \). This can make convergence and stability of the optimization to be more problematic, which has also been observed in prior work [25]. A similar problem occurs for UOI when the network tends to predict that everything is background. To help avoid this behavior, in our implementation, we first optimize the network using a softmax loss in order to ensure that there are sufficiently many foreground predictions. We then further optimize the network using the UOI loss.

\subsection*{5.4.2.2 DATA GENERATION}

As mentioned earlier, since our CNN sub-network depends on the output mask predicted for the previous frame, the training data is no longer independent. Moreover, for lots of segmentation benchmarks, ground truth is not available for every frame which eliminates the option to use ground truth as the history input in such datasets. To overcome these problems, we propose a DAGGER style training approach [84] to train our CNN.

The original DAGGER algorithm was developed for imitation learning of controllers/policies for sequential decision making problems in Markov Decision Processes. The key idea behind imitation learning is to learn to perform a task by learning to imitate an expert at the task, who is available at training time. In the case of tracking-by-segmentation, we can treat ground-truth
annotations as the expert and attempt to “imitate” those annotations. To simplify our description, we will describe DAgGER in the context of tracking-by-segmentation, noting that it applies much more broadly. For more details about the general DAgGER approach and its theoretical guarantees, please refer to [84].

We first describe DAgGER under the assumption that all frames have annotations. DAgGER is an iterative algorithm, which produces a sequence of trackers learned on a progressively increasing set of training examples. In the first iteration, we construct training examples using the expert (i.e. ground truth masks). More specifically, a labeled training sample is created for each frame $t$, where the input to the network for the example is a triplet $(I_t, I_{t-1}, M_{t-1})$ and the target output is $M_t$. Here $I_t$ and $M_t$ is the image and ground truth mask at time $t$ respectively. This gives us a initial training set $D_1$, which is used to train our CNN, which we denote by $CNN_1$. In iteration $k + 1$ of DAgGER, we first run the previously learned tracker, $CNN_k$, on the training sequences to get the predicted mask for every frame. Denote these masks as $\hat{M}_k^t$. A new training set $D^{k+1}$ is then created with the input of each example taking the form of $(I_t, I_{t-1}, \hat{M}_{t-1}^k)$ and again using the ground truth as the target output. We add $D^{k+1}$ to the growing set of training data and use the new data to finetune our CNN based on $CNN_k$ and obtain a new model $CNN_{k+1}$. This process is repeated for a fixed number of iterations. Note that the above process requires the ground truth masks for every frame in the first iteration. A pseudocode of the algorithm can be seen at Alg. 15. The key insight behind DAgGER is that by using the learned CNNs to generated training data, we are learning on situations that will be encountered in practice at test time. In contrast, only learning on expert trajectories can lead to poor predictions at test time when the learned CNN departs from the expert.

For datasets without annotations for each frame, we cannot directly use DAgGER, since it requires that the expert prediction be available for each frame. To address this problem, we first trained a CNN model on another dataset that does have full annotations in order to arrive at an initial tracking CNN. We then use that tracker to generate predicted masks in each frame. For frames where we have ground truth annotations, we can form training examples as above using the predicted masks to form the input data of the training examples and the ground truth as the target value. However, this does not produce training data for frames without ground truth annotations, since it is unclear what to use for the target value of those examples. To supplement the training data we follow a recent observation [57] that adding noisy data can help obtain better test performance in some situations. To do this, suppose that we have ground truth masks for frames $t$ and $t + \delta$. We fit a tight bounding box to these two masks and a linear interpolation is
done to get the bounding boxes for frames in between \( t \) and \( t + \delta \). We also add small random perturbations to the interpolated locations. For frames from \( t + 1 \) to \( t + \delta / 2 \), we translate the mask for frame \( t \) to the corresponding bounding box location, and for frames \( t + \delta / 2 + 1 \) to \( t + \delta \), we use frame \( t + \delta \). These generated masks will be used as the target output for frames with no provided ground truth annotations.

**Algorithm 3**

Our CNN training algorithm for one video

**Input:**
- A set of \( T \) frames \( \mathbb{I}_t \)
  - Ground truth masks for each frame \( \mathbb{M}_t \)
  - A fixed number of iterations \( N \)

**Output:**
- A CNN model \( CNN_N \)

1: Set initial training set \( D^1 \) to \( \emptyset \)
2: for \( t \leftarrow 2 \) to \( T \) do
3:   add a training pair \((\mathbb{I}_t, \mathbb{I}_{t-1}, \mathbb{M}_{t-1}), \mathbb{M}_t)\) to \( D_1 \)
4: end for
5: \( CNN_1 = \text{TRAINCNN}(D_1) \)
6: for \( k \leftarrow 1 \) to \( N - 1 \) do
7:   \( \mathbb{M}_k^k = \mathbb{M}_1 \)
8:   Set training set \( D^{k+1} \) to \( \emptyset \)
9:   for \( t \leftarrow 2 \) to \( T \) do
10:      \( \mathbb{M}_k^k = \text{RUNCNN}(CNN_k, (\mathbb{I}_t, \mathbb{I}_{t-1}, \mathbb{M}_{k-1})) \)
11:     add a training pair \((\mathbb{I}_t, \mathbb{I}_{t-1}, \mathbb{M}_k^k), \mathbb{M}_t)\) to \( D_{k+1} \)
12:   end for
13: \( CNN_{k+1} = \text{FINETUNECNN}(D_{k+1}) \)
14: end for
15: Return \( CNN_N \)

### 5.5 EXPERIMENTAL RESULTS

**Datasets** We evaluate our method on two datasets. 1) The SegTrackV2 dataset[62] contains 14 videos with 24 objects. Ground truth annotation is provided for every frame. 2) The Youtube dataset [80] contains 126 videos from 10 different classes. Ground truth annotations are provided by [48] for approximately every 10 frames. Challenges in both datasets include object interactions, appearance changes and complex deformation.

**Implementation Details.** We implemented our CNN+RNN model using CAFFE[51]. For
the SegTrackV2 dataset, we use leave one video out to evaluate our network. For the Youtube dataset, which does not contain annotations for each frame, we first train a CNN on the entire SegTrackV2 dataset to initialize our DAGGER approach as described above that is trained on the Youtube data. The learning rate is set to $10^{-7}$ for CNN training and $10^{-3}$ for the RNN. For each dataset, the CNN is first trained for three iterations using our data aggregation algorithm. The RNN is then trained with the CNN outputs. We further finetune the two networks jointly using an additional two DAGGER iterations. A post-process is done based on the network outputs where a simple pairwise CRF is built upon SLIC superpixels [2] to further smooth the results. The edge image we use for the RNN input comes from [29].

**Baselines and Evaluation Metrics** For both datasets we use intersection over union (IOU) as our evaluation metrics. Along with our full model, we also evaluate the effectiveness of each module. Specifically, we denote *Ours* to be our full CNN+RNN model. $CNN_i$ denotes the CNN trained after the $i$th DAGGER iteration without the RNN refinement model. For the Youtube dataset, we also evaluate the effectiveness of the created noisy ground truth mask. We use Ours-less to denote models that are trained without these data—that is, models trained only using the annotated video frames. On the SegTrackv2 dataset, we compare with traditional tracking by segmentation methods [99, 98] and video segmentation methods [30, 53, 87]. On the Youtube dataset, we compare to both video segmentation and deep semantic segmentation networks [30, 53, 87, 93].

**Quantitative Results.** Table 5.3 shows the comparison to our baselines on the two datasets. We can see that the biggest improvement in both datasets comes from going from $CNN_1$ to $CNN_2$, which correspond to the first and second iteration of DAGGER training. Recall that when training $CNN_2$ in the second DAGGER iteration, we add the training data based on the learned $CNN_1$ trajectories. This shows that augmenting training data with trajectories from learned policies can lead to significant improvements. Recall that for SegtrackV2, $CNN_1$ is trained on the ground truth history while in Youtube, $CNN_1$ is trained on the predicted mask of a network trained on SegtrackV2. The second contribution comes from adding the RNN layer (the improvement from $CNN_3$ to Ours) which demonstrate the utility of refining the mask with an edge image. Comparing Ours to Ours-less, we can see that the generated noisy annotations also helps improve the results.

Table 5.1 compares our method to other state-of-the-art approaches on the SegTrackV2 dataset. We can see that our approach outperforms all competitors in terms of average IOU.
The closest competitor [30] performs very well on some sequences and completely fails on others. This is because the approach in [30] is based on a trained deep network for detecting particular object classes. When the sequence contains the object class that the detector is trained on, [30] is able to achieve good performance, but otherwise it can perform poorly. Compared to the traditional (non-deep) tracking-by-segmentation methods [19, 98], our network performs consistently better. Note that we are aware of a traditional tracking-by-segmentation work [99] that reports an average performance of 71.8. However, an inspection of their code suggests that their evaluation uses different hyperparameters for different sequences, which leads to an unfair comparison with other approaches that use a fixed set of parameters that are not tuned per sequence.

Table 5.2 shows the results for the Youtube dataset. Compared to the state-of-the-art approach, we can see that our approach outperform the deep semantic segmentation network [87] by quite a margin. Our results are slightly worse than [30]. The primary reason for this is that there are two scenarios in the Youtube dataset that our method is not able to handle. One is that in some videos, the camera viewpoint changes, which causes our approach to lose the target. Second, in some videos, a new object instance appears in the middle of the video. Since our method only tracks and segments whatever is given in the first frame, we are not able to capture this situation. In both situations, approaches such as [30] that are based on object classes have an advantage when the videos in question involve the set of object classes covered by the network.

Qualitative evaluation: Fig. 5.3 and Fig. 5.4 show our results on several videos in the SegTrackV2 and Youtube dataset respectively. We can see that our network captures most part of the target but still has some problems with the boundaries especially with some articulated parts like the arms of the girl/monkey. Fig. 5.4 also shows the results for a traditional tracking-by-segmentation algorithm [99]. We can see [99] fails in all these cases. This is mainly because it is hard to find uniform hyperparameters that work well across all sequences and that the online appearance model update can easily drift to non-target parts of the video.

5.6 CONCLUSION

We presented a new deep architecture for online tracking-by-segmentation, which can be used to track and segment an object throughout a video. To the best of our knowledge, this is the first attempt to have a full feed-forward network for tracking-by-segmentation of a generic object. Our network consists of a CNN that given the current frame, previous frame and previous mask,
predicts a coarse mask of the target in current frame. The coarse mask is then refined iteratively via a RNN. A key aspect of our approach is that the input to the tracker depends on the previously predicted tracking result, which raises several challenges in training the network. To solve these challenges, we proposed an iterative training algorithm that aggregates data from multiple training runs of the tracker. We also proposed a data augmentation approach that generates noisy ground truth masks for frames that are not annotated with ground truth masks. Experimental results on the SegtrackV2 and Youtube dataset shows our network is able to achieve competitive or better results compared to state-of-the-art alternatives.
Figure 5.1: Our CNN architecture. Note that we remove the pooling layers which is widely used in other CNNs. We empirically found that this change improves results for our problem. Such non-pooling networks have also been used for saliency detection such as [63].

<table>
<thead>
<tr>
<th>Layer</th>
<th>Filter Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (320<em>240</em>7)</td>
<td></td>
</tr>
<tr>
<td>Convolution 1 (7<em>7</em>96)</td>
<td></td>
</tr>
<tr>
<td>Local Response Normalization</td>
<td></td>
</tr>
<tr>
<td>Convolution 2 (5<em>5</em>256)</td>
<td></td>
</tr>
<tr>
<td>Convolution 3 (3<em>3</em>512)</td>
<td></td>
</tr>
<tr>
<td>Convolution 4 (5<em>5</em>512)</td>
<td></td>
</tr>
<tr>
<td>Convolution 5 (5<em>5</em>512)</td>
<td></td>
</tr>
<tr>
<td>Convolution 6 (7<em>7</em>256)</td>
<td></td>
</tr>
<tr>
<td>Convolution 7 (7<em>7</em>2)</td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.2: Our unrolled RNN. $m^i_t$ denotes the mask from $i$th iteration where the initial mask $m^0_t$ comes from our CNN. $\Delta^i_t$ denotes the modification the module outputs in the $i$th iteration. $e_t$ is the edge image we extract in advance to help improve the boundary.
<table>
<thead>
<tr>
<th>Seq</th>
<th>Ours</th>
<th>[30]</th>
<th>[53]</th>
<th>[87]</th>
<th>[19]</th>
<th>[98]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Deep</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>bird-of-paradise</td>
<td>48.3</td>
<td>50.5</td>
<td>79.0</td>
<td>43.2</td>
<td>46.5</td>
<td>44.3</td>
</tr>
<tr>
<td>birdfall</td>
<td>32.6</td>
<td>4.5</td>
<td>0.5</td>
<td>-</td>
<td>36.4</td>
<td>32.5</td>
</tr>
<tr>
<td>bmx-person</td>
<td>84.9</td>
<td>90.7</td>
<td>70.4</td>
<td>0.9</td>
<td>36.0</td>
<td>27.9</td>
</tr>
<tr>
<td>bmx-bike</td>
<td>7.24</td>
<td>33.5</td>
<td>17.3</td>
<td>20.0</td>
<td>3.86</td>
<td>6.04</td>
</tr>
<tr>
<td>cheetah-deer</td>
<td>40.6</td>
<td>41.9</td>
<td>1.9</td>
<td>-</td>
<td>38.7</td>
<td>33.1</td>
</tr>
<tr>
<td>cheetah-heetah</td>
<td>19.4</td>
<td>0</td>
<td>3.9</td>
<td>-</td>
<td>19.7</td>
<td>14.0</td>
</tr>
<tr>
<td>drift-car1</td>
<td>58.4</td>
<td>70.1</td>
<td>50.2</td>
<td>36.0</td>
<td>57.2</td>
<td>43.5</td>
</tr>
<tr>
<td>drift-car2</td>
<td>49.3</td>
<td>60.2</td>
<td>0.3</td>
<td>39.6</td>
<td>13.8</td>
<td>11.6</td>
</tr>
<tr>
<td>frog</td>
<td>51.6</td>
<td>68.4</td>
<td>43.1</td>
<td>-</td>
<td>38.8</td>
<td>45.2</td>
</tr>
<tr>
<td>girl</td>
<td>78.3</td>
<td>65.4</td>
<td>51.4</td>
<td>65.8</td>
<td>62.0</td>
<td>52.4</td>
</tr>
<tr>
<td>monkey</td>
<td>65.7</td>
<td>65.2</td>
<td>22.8</td>
<td>-</td>
<td>58.7</td>
<td>61.7</td>
</tr>
<tr>
<td>monkeydog-d</td>
<td>10.9</td>
<td>0</td>
<td>6.8</td>
<td>0</td>
<td>3.83</td>
<td>10.2</td>
</tr>
<tr>
<td>monkeydog-m</td>
<td>67.3</td>
<td>77.7</td>
<td>54.9</td>
<td>0</td>
<td>25.7</td>
<td>22.1</td>
</tr>
<tr>
<td>hummingbird-1</td>
<td>42.8</td>
<td>10.4</td>
<td>11.0</td>
<td>39.8</td>
<td>25.1</td>
<td>28.8</td>
</tr>
<tr>
<td>hummingbird-2</td>
<td>38.6</td>
<td>9.4</td>
<td>32.4</td>
<td>30.2</td>
<td>44.2</td>
<td>45.9</td>
</tr>
<tr>
<td>parachute</td>
<td>52.9</td>
<td>71.9</td>
<td>89.8</td>
<td>-</td>
<td>59.3</td>
<td>69.9</td>
</tr>
<tr>
<td>penguin-1</td>
<td>72.3</td>
<td>43.0</td>
<td>8.6</td>
<td>20.0</td>
<td>40.1</td>
<td>20.8</td>
</tr>
<tr>
<td>penguin-2</td>
<td>68.4</td>
<td>0</td>
<td>4.1</td>
<td>0.7</td>
<td>37.9</td>
<td>20.8</td>
</tr>
<tr>
<td>penguin-3</td>
<td>66.1</td>
<td>0</td>
<td>3.8</td>
<td>14.9</td>
<td>31.2</td>
<td>10.3</td>
</tr>
<tr>
<td>penguin-4</td>
<td>57.9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30.2</td>
<td>13.0</td>
</tr>
<tr>
<td>penguin-5</td>
<td>57.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10.7</td>
<td>18.9</td>
</tr>
<tr>
<td>penguin-6</td>
<td>58.8</td>
<td>73.6</td>
<td>5.3</td>
<td>0.61</td>
<td>35.0</td>
<td>32.3</td>
</tr>
<tr>
<td>soldier</td>
<td>63.4</td>
<td>64.0</td>
<td>50.1</td>
<td>0</td>
<td>54.2</td>
<td>43.0</td>
</tr>
<tr>
<td>worm</td>
<td>65.6</td>
<td>69.0</td>
<td>23.0</td>
<td>-</td>
<td>44.3</td>
<td>27.4</td>
</tr>
<tr>
<td>aver obj</td>
<td>52.4</td>
<td>48.0</td>
<td>34.5</td>
<td>24.9</td>
<td>35.6</td>
<td>30.7</td>
</tr>
<tr>
<td>aver vid</td>
<td>51.8</td>
<td>40.3</td>
<td>26.3</td>
<td>18.3</td>
<td>40.4</td>
<td>37.0</td>
</tr>
</tbody>
</table>

Table 5.1: Results for the SegTrackv2 dataset. We describe each method with two attributes. Generic means that the method is not trained for specific class and can be applied to generic object. Deep indicates whether the method ever uses any deep networks.
Table 5.2: Results for the Youtube dataset.

<table>
<thead>
<tr>
<th>Seq</th>
<th>Ours</th>
<th>[30]</th>
<th>[77]</th>
<th>[87]</th>
<th>[93]</th>
</tr>
</thead>
<tbody>
<tr>
<td>aero</td>
<td>75.3</td>
<td>74.4</td>
<td>65.5</td>
<td>59.3</td>
<td>17.8</td>
</tr>
<tr>
<td>bird</td>
<td>68.3</td>
<td>72.1</td>
<td>69.2</td>
<td>67.6</td>
<td>19.8</td>
</tr>
<tr>
<td>boat</td>
<td>56.1</td>
<td>58.5</td>
<td>43.9</td>
<td>32.6</td>
<td>22.5</td>
</tr>
<tr>
<td>car</td>
<td>52.7</td>
<td>60.0</td>
<td>66.1</td>
<td>50.5</td>
<td>38.3</td>
</tr>
<tr>
<td>cat</td>
<td>43.9</td>
<td>45.7</td>
<td>49.3</td>
<td>33.1</td>
<td>23.6</td>
</tr>
<tr>
<td>cow</td>
<td>59.9</td>
<td>61.2</td>
<td>38.0</td>
<td>27.4</td>
<td>26.8</td>
</tr>
<tr>
<td>dog</td>
<td>55.8</td>
<td>55.2</td>
<td>50.0</td>
<td>35.6</td>
<td>23.7</td>
</tr>
<tr>
<td>horse</td>
<td>49.7</td>
<td>56.6</td>
<td>31.6</td>
<td>46.0</td>
<td>14.0</td>
</tr>
<tr>
<td>motorbike</td>
<td>40.9</td>
<td>42.1</td>
<td>31.9</td>
<td>18.4</td>
<td>12.5</td>
</tr>
<tr>
<td>avg cat</td>
<td>55.8</td>
<td>56.2</td>
<td>47.9</td>
<td>41.8</td>
<td>23.9</td>
</tr>
<tr>
<td>avg vid</td>
<td>54.4</td>
<td>55.8</td>
<td>45.7</td>
<td>37.3</td>
<td>22.8</td>
</tr>
</tbody>
</table>

Table 5.3: Results compared with baselines

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SegTrackV2</th>
<th>Youtube</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CNN_1$</td>
<td>48.7</td>
<td>49.3</td>
</tr>
<tr>
<td>$CNN_2$</td>
<td>50.4</td>
<td>52.9</td>
</tr>
<tr>
<td>$CNN_3$</td>
<td>51.3</td>
<td>54.1</td>
</tr>
<tr>
<td>Ours-less</td>
<td>-</td>
<td>55.1</td>
</tr>
<tr>
<td>Ours</td>
<td>52.4</td>
<td>55.8</td>
</tr>
</tbody>
</table>
Figure 5.3: Results on SegTrackV2 dataset
Figure 5.4: Results on Youtube dataset. For each sequence, the second row shows our results and the third row shows results of [99]
Chapter 6: Conclusion and Future Directions

In this thesis, we mainly focus on object tracking in videos. While most existing approaches represent targets using bounding boxes, we instead use segments as they allow us to better handle occlusions, interactions and also give better access to feature descriptions. We propose three methods. First, we developed the constrained sequential labeling algorithm that given an over-segmentation of the video, sequentially labels these over-segmented features with a cost function and a set of hard constraints. Second, we proposed a novel error-driven graph revision framework that given a noisy under-segmentation and object detections, outputs a set of detections (segments/bounding boxes) and counts the number of targets inside each detection which can be used for tracking. Finally, we proposed a deep network architecture that provides an end-to-end solution to the problem of tracking-by-segmentation.

Tracking is a very difficult task in general. While tracking-by-segmentation approaches avoid many problems brought by bounding box based representations, it is also a generally harder problem since the solution space is much larger. Traditional tracking-by-segmentation algorithms usually suffer from three problems. First, many algorithms builds on certain assumptions. For example, our CSL work assumes a good over-segmentation of the video is available. Our count localization algorithm is very likely to fail if the video contains only one large group moving together. Second, most algorithms are sensitive to certain hyperparameters like [99, 106] and thus may not work well on an arbitrary video. Third, most tracking-by-segmentation algorithm are very slow and cannot run in real time.

The development of deep learning in recent years has demonstrated its power in solving large-scale vision problems. However, unlike image classification where deep networks beat traditional methods by quite a large margin and can even achieve super-human performance, in segmentation and tracking, the improvement over traditional methods are moderate and there is still much room to improve both quantitatively and qualitatively. While this thesis has developed the first attempt to build a deep network for tracking-by-segmentation, the results are still not qualitatively impressive and robust, despite achieving close to state-of-the-art performance. In particular, the network suffers in certain conditions such as new targets appearing in the middle of the video.
Very recently, [49] proposed a two-stream network that segments generic objects in videos. However, since the network is initially trained on classification tasks and then finetuned with segmentation masks, the generality of this network is unclear. Ultimately some combination of class specific [30] and class agnostic (ours) approaches is needed to handle new classes while also exploiting knowledge of known classes.
Bibliography


