Multi-Modal Models for Fine-grained Action Segmentation in Situated Environments

by

Colin Lea

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

March, 2017

Abstract

Automated methods for analyzing human activities from video or sensor data are critical for enabling new applications in human-robot interaction, surgical data modeling, video summarization, and beyond. Despite decades of research in the fields of robotics and computer vision, current approaches are inadequate for modeling complex activities outside of constrained environments or without using heavily instrumented sensor suites. In this dissertation, I address the problem of fine-grained action segmentation by developing solutions that generalize from domain-specific to general-purpose for applications in surgical workflow, surveillance, and cooking.

A key technical challenge, which is central to this dissertation, is how to capture complex temporal patterns from sensor data. For a given task, users may perform the same action at different speeds or styles, and each user may carry out actions in a different order. I present a series of temporal models that address these modes of variability. First, I define the notion of a convolutional action primitive, which captures how low-level sensor signals change as a function of the action a user is performing. Second, I generalize this idea to video with a Spatiotemporal Convolutional Neural

ABSTRACT

Network, which captures relationships between objects in an image and how they change temporally. Lastly, I discuss a hierarchical variant that applies to video or sensor data, called a Temporal Convolutional Network (TCN), which models actions at multiple temporal scales. In certain domains (e.g., surgical training), TCNs can be used to successfully bridge the gap in performance between domain-specific and general-purpose solutions.

A key scientific challenge concerns the evaluation of predicted action segmentations. In many applications, action labels may be ill-defined and if one asks two different annotators when a given action starts and stops they may give answers that are seconds apart. I argue that the standard action segmentation metrics are insufficient for evaluating real-world segmentation performance and propose two alternatives. Qualitatively, these metrics are better at capturing the efficacy of models in the described applications.

I conclude with a case-study on surgical workflow analysis, which has the potential to improve surgical education and operating room efficiency. Current work almost exclusively relies on extensive instrumentation, which is difficult and costly to acquire. I show that our spatiotemporal video models are capable of capturing important surgical attributes (e.g., organs, tools) and achieve state-of-the-art performance on two challenging datasets.

The models and methodology described have demonstrably improved the ability to temporally segment complex human activities, in many cases without sophisticated

ABSTRACT

instrumentation.

Primary Advisor: Gregory D. Hager (Johns Hopkins University)

Co-Advisor: René Vidal (Johns Hopkins University)

Co-Advisor: Austin Reiter (Johns Hopkins University)

Reader: Nicolas Padoy (University of Strasbourg)

Acknowledgments

Years ago, as a prospective PhD student, I visited several world-class universities. Each school had wonderful faculty, excellent research opportunities, and room for growth, but there was one thing that truly stood out about Johns Hopkins. From my one visit I could tell their was a sense of community unlike at any of the others. The open environment within LCSR and the interactions I saw between students left a lasting impression. Five and a half years later, I still find this to be true. I have had great opportunities working side-by-side with faculty and students in multiple departments, unique experiences interacting with clinicians in the school of medicine, and rewarding collaborations with universities in the US and in Germany. I could not be happier with my decision to do my PhD at Johns Hopkins.

First, I would like to thank Greg Hager, my primary advisor. Greg has an uncanny ability to see the big picture even when presented with a thousand layers of minutiae. He has directly enabled numerous opportunities for me over the years, ranging from collaborations with colleagues in Munich and at Berkeley to clinical collaborations with the medical school. All of these experiences have had an important impact on

this dissertation and on the kind of researcher that I have become. Greg has been nothing but supportive; giving me independence when I need it and pushing me when I am stuck.

I would like to thank my co-advisors Rene Vidal and Austin Reiter. Rene has a razer sharp mathematical mind and has instilled a sense of rigor in me beyond what is common in the literature. Because of him, I am skeptical of all papers, peer-reviewed or not, and need to double-check for myself that presented claims are indeed correct. Rene's strong paper editing skills have enabled me to be a more detailed and technical writer.

Austin's eye for practicality has been crucial for our research on surgical workflow. Over the years, Austin and I have had dozens of fruitful meetings and conversations, but there is one that had the largest impact on my thesis work. We started the meeting talking about surgical results, but by the end had unintentionally laid out what would ultimately became the Temporal Convolution Network, which forms the capstone to this dissertation.

I want to thank Nicolas Padoy, my outside committee member. When I started, Nicolas was a research faculty in LCSR working on computer vision and surgical robotics. A year later he left to start what is becoming an incredible research group at IRCAD in Strasbourg. Earlier this year, I was excited to learn that his group was working on some of the same problems as us in the area of surgical workflow analysis. I am grateful for his presence on my committee.

ACKNOWLEDGMENTS

Early in my PhD I had the pleasure to work with Russ Taylor, Suchi Saria and collaborator Jim Fackler on an incredibly rewarding project applying computer vision to clinical problems in Intensive Care Units. It was really this project that got me started in the fascinating realm of human activity analysis.

Thanks Sanjeev Khudanpur and Daniel Robinson who were on my GBO exam committee. They pushed me to be my best and to improve my mathematical background.

I want to thank my close student collaborators. From robotics (Kel Guerin, Jon Bohren, Chris Paxton, Chi Li, Christian Rupprecht), Language of Surgery (Swaroop Vedula, Narges Ahmidi, Anand Malpani, Lingling Tao, Yixin Gao, Rob DiPietro), computer vision (Lingling, Siddharth Mahendran, Efi Mavroudi), to various other projects that I have touched (Will Gray-Roncal; Animesh Garb and Sanjay Krishnan at Berkeley). I also want to thank the masters and undergrad students who have assisted over the years. Notably, Flynn Flynn who was invaluable in some of my recent work and James Choi who helped with much of the surgical workflow project.

Of course my PhD experience would not be the same without everyone else in LCSR. This list is too big to enumerate, but I would be remissed if I did not mention Alison Morrow, Lorrie Dodd, Rose Chase, Julia Ortiz-Foy, and everyone else who really makes LCSR operate. The same can be said of the computer science department. Thanks to Tracy Marshall, Debbie Deford, Zack Burwell, Laura Graham, and the rest of the administrative staff for making my life easier.

ACKNOWLEDGMENTS

Lastly, I had the great experience of leading the JHU Robo Challenge outreach event for several years. This would not have been possible without the great help of Anita Sampath, Xinchi He, Josh Davis, Andrew Spielvogel, and all of our volunteers. I am grateful that Andrew has taken the reigns and is continuing to make this program a success.

Every PhD has its ups and downs, but despite the hard times I cannot imagine doing my PhD anywhere else. I am truly going to miss Hopkins.

Dedication

To my parents Kathy and Doug, my brother Keith and his fiancée Sam, and my fiancée Emily.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The ability to model how humans interact with their environment is crucial for developing many next generation assistive technologies. This may require modeling human motions, objects in their environment, and the interactions between them. In a manufacturing context, the ability to detect when a human has completed a complex assembly may enable collaborative robots to assist with mundane tasks like part fetching [11]. During robotic surgery training, detecting when a trainee has completed a suture throw, or when they accidentally drop a needle, may enable quantitative and automatic skills assessment. In an age when humans are becoming more and more comfortable interacting with smart environments, there is a growing need to develop methods that enable sophisticated interactions between them.

In this thesis, we address the problem of fine-grained action segmentation in sit-

uated environments from video or sensor data. In this context, *situated* refers to a specific setting, such as a manufacturing setup or surgical training station, and *fine-grained* refers to actions that are subtly different from one another. Two example actions in a cooking task may be `cutting a carrot` and `peeling a cucumber`. Our goal is to take a recording of a user performing a given activity and temporally segment each of the constituent actions.

While there has been substantial body of work addressing action segmentation, as we will describe in Chapter 2, many ongoing challenges have prevented its use in real-world applications. Roboticists often develop domain-specific solutions that offer compelling performance in well-calibrated, instrumented scenes, but which do not generalize to new environments. Furthermore, features are often hand-crafted specifically to model a small set of actions, and require additional work to model new actions. Computer vision scientists often develop very general solutions that work in a wide variety of domains, but whose performance is insufficient for use in practice. Challenges arise due to variation in object appearance, subtle hand motions, and failure to model the relationships between a human and objects in the scene and how they change over time. In this thesis, we attempt to bridge the gap between accurate but domain-specific solutions and inaccurate but general-purpose solutions. Through our development of multi-modal spatiotemporal models and hierarchical time-series models, we show that in some cases we do in fact close this performance gap.

While this thesis comes out of work modeling activities in several specific domains

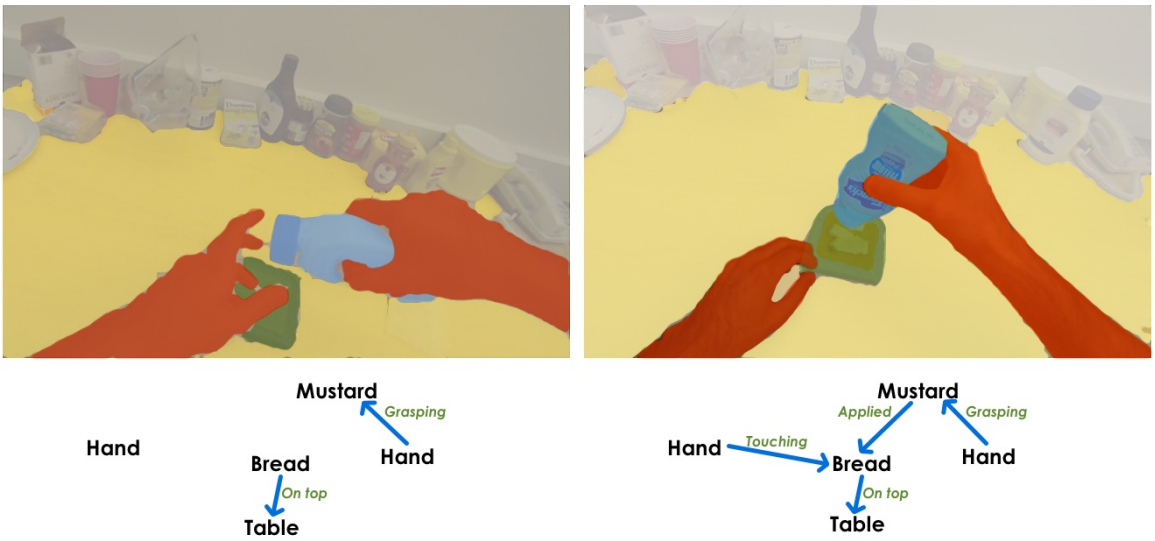


Figure 1.1: In this thesis we develop models that capture what objects are in an image, their spatial relationships, and how these relationships change over time. The top images highlight the relevant objects and the bottom is represents the relevant relationships. We capture these properties via latent layers in a Convolutional Neural Network. These are annotated using images from the Georgia Tech Egocentric Activities dataset [3].

– as described in Section 1.1 – the goal is to develop generalizable methods that are effective in a wide range of applications. Throughout this thesis we describe results on over a half-dozen datasets, which address applications in human-robot interaction, video summarization, surveillance, surgical training, and surgical workflow analysis.

1.1 Motivating Applications

While the technical contributions in this thesis will be described in the context of publicly available datasets, for example using the cooking videos in 50 Salads [12], many of the ideas are a direct result of progress on several applications not typically

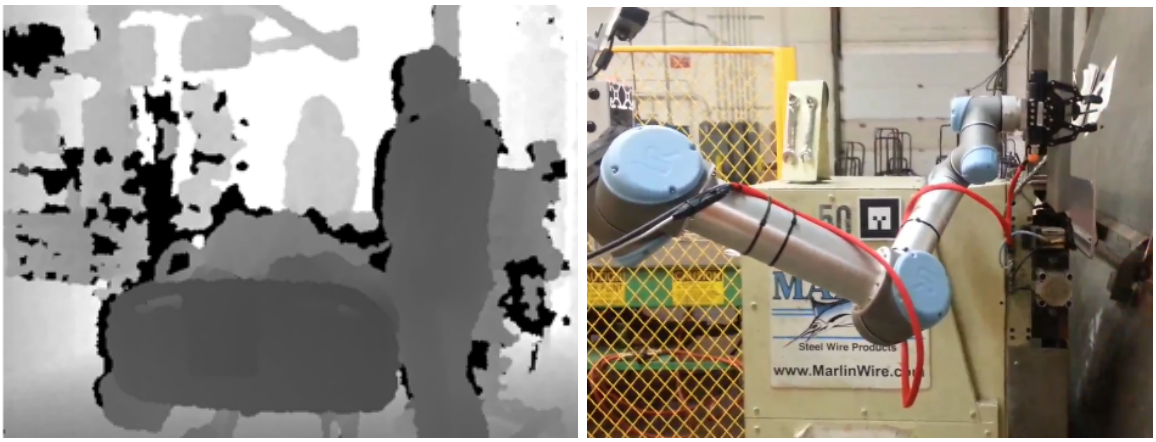


Figure 1.2: Motivating applications: (left) a depth image recorded in an intensive care unit (right) our robot being deployed at a small-batch manufacturer.

used by the action recognition community. My experiences working on these applications helped highlight many facets of activity recognition that are not common in the literature, but which are important for bringing the technology out of the lab and into the real-world.

In the following subsections I will briefly describe applications. We will discuss two of these – surgical skill assessment and surgical workflow analysis – in more depth later.

1.1.1 Automated Monitoring in Intensive Care Units

ICUs are chaotic places where hundreds of tasks are carried out by many different people. Proper execution of these tasks, in a timely manner, is important for ensuring quality of patient outcomes. The goal of our work [13, 14] was to prototype a system that could automatically catalog various tasks performed by the bedside using

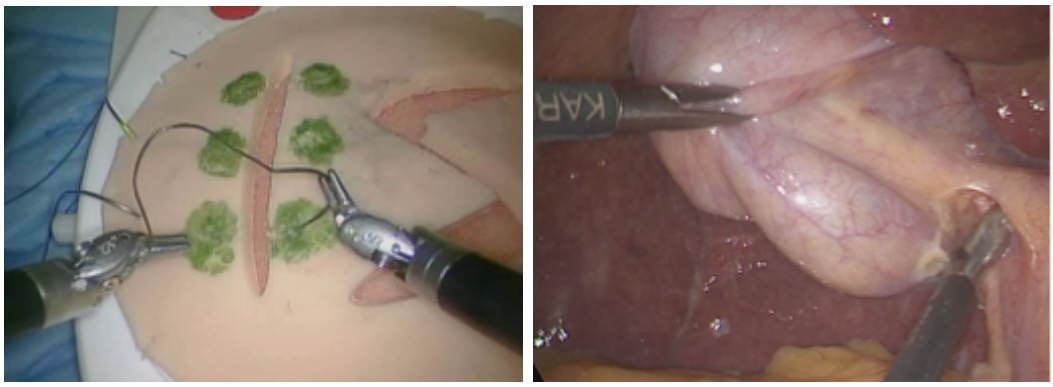


Figure 1.3: Motivating applications: (left) an image from a suturing task for surgical skill assessment (right) an image of a Cholecystectomy used for surgical workflow analysis.

passive RGBD-based sensing. We proposed a set of computer vision and machine learning techniques to develop a system that identified seven common actions such as documenting, checking up on a patient, and performing a procedure. Our system showed promising preliminary results for recognizing tasks from data we collected at a Pediatric ICU Furthermore, we showed how this technology could be used to improve understanding of the current care – by summarizing and visualizing task completion – to improve overall quality. This system was a significant departure from prior approaches for quality improvement, which typically required that nurses spend excessive amounts of time manually entering data about each patient.

1.1.2 Surgical Skill Assessment

The goal of our work in [15, 16] was to advanced the state of the art in surgical skill assessment through the development of action segmentation models. In recent years there have been many calls for improving the quality and efficacy of training for

robotic surgery [17, 18]. Current methods for skill evaluation in surgical training tasks tend to be either too subjective or too time consuming, and our hypothesis is that automated, quantitative methods may reduce common issues including inter-reviewer variability and inter-personal bias. We used sensor data collected from daVinci surgical robots, including video and robot kinematics (e.g. position and velocity of the robot end effectors), to recognize the sequence of actions a surgeon took to perform a robotic surgery training task. While our contributions are targeted at recognizing the individual actions, these can in-turn be used to evaluate the skill of a user. We use JIGSAWS [19], a surgical training dataset developed by earlier members of our group, as a running example throughout the primary technical chapters of this thesis. In a recent collaboration we evaluated our models on a more recent surgical training dataset, MISTIC [16], however this is outside of the scope of this thesis.

1.1.3 Surgical Workflow Analysis

Most recently, we worked towards surgical phase analysis from videos of robotic and laparoscopy procedures including cholecystectomies [20] and hysterectomies [21]. Automatic segmentation of laparoscopic recordings into sequences of clips is important for analyzing workflow, improving surgical education, and providing surgeons with automated feedback. Despite increasing interest in this problem, current work almost exclusively relies on extensive instrumentation, which is difficult and costly to acquire, and is only evaluated on data collected from individual institutions. In this

work we found two methodological advances for video-based surgical phase segmentation at-large, and additionally, introduced a new multi-institution surgical phase segmentation dataset. We employed video-based, sensor-based, and multi-modal models on this data to achieve state-of-the-art performance on the TUM EndoVis dataset which includes a small number of cholecystectomy procedures. We also applied this work to datasets our group collected for hysterectomies, cataracts surgery, and to the M2CAI 2016 Surgical Workflow Competition[1]. Lastly, we curated a dataset from cholecystectomy videos that were performed around the world to assess the ability of our models to generalize to a broader variety of surgical styles, equipment types, and lighting conditions.

1.1.4 Collaborative Robotics

In [11] we addressed problems with using collaborative robots in Small Manufacturing Entities (SMEs). This project highlighted the importance of modeling actions as a function of changes in the workspace, including changes in object position or state. Historically, SMEs have not incorporated robotic automation as readily as large companies due to rapidly changing product lines, complex and dexterous tasks, and the high cost of start-up. In this work we designed a platform, CoSTAR (Collaborative System for Task Automation and Recognition) that incorporated easy-to-use robot instruction, object perception, and human action modeling. While the system

[1]M2CAI Surgical Workflow Competition `camma.u-strasbg.fr/m2cai2016/`

for action recognition was relatively simple, it highlighted important challenges for modeling human actions in complex environments.

1.2 Outline and Approaches

The primary focus of this work is on the development of generalizable models for action segmentation that can be applied across many unique environments, including those just described. In particular, we work towards bridging the gap between solutions from the robotics community, whose results are promising but typically require that an environment be instrumented with costly or invasive sensors (e.g. [22, 5]), and solutions from the computer vision community, which typically only require a camera but whose performance is more modest and likely unusable for the real-world (e.g. [23, 24]). A detailed description of each chapter is included below. Before describing our work, in Chapter 2 we highlight the state of the art in robotics and computer vision.

The first technical contributions appear in Chapter 3 where we develop a baseline time-series model for action segmentation that achieves state-of-the-art performance on two public datasets using domain-specific sensors. Our Skip Chain Conditional Random Field is an efficient higher-order model that captures high-level temporal dynamics, which we find outperforms Linear Chain CRFs, Hidden Markov Models, Recurrent Neural Networks, and other time-series models. We introduce the vanilla Skip Chain CRF, a latent variant, and a segmental variant. We combine these CRFs

with our first notion of a temporal action primitive. These action primitives model how sensor values change over time, and in contrast with many hand-crafted time-series features (e.g.[4, 22, 25, 5, 12]), are learned directly from raw sensor data.

We then introduce a spatiotemporal model in Chapter 4 that, in tandem with the time-series model, improves action segmentation performance when operating solely on video. The spatial component of this model captures the relationships between different objects within each image and the temporal component captures how these relationships change across time. We investigate geometric relationships between objects by exploiting the fact that in many applications we consider, the camera is typically stationary. By contrast, the state-of-the-art approaches in video-based action segmentation use holistic representations that ignore spatial structure. The second contribution in this chapter is a method for training spatial CNNs that does not require any training labels. We use sensor data that is synchronized with the video to learn a video-based representation that can the be used for action segmentation or other tasks. This is especially important in robotics applications where there is a large amount of training data without ground truth labels but for which there is auxiliary sensor data.

Finally in Chapter 5 we introduce a class of hierarchical, fully convolutional models, called Temporal Convolutional Networks (TCN), that unify ideas from our time-series baseline and spatiotemporal representation to, in some cases, fully bridge the gap between solutions using domain-specific sensing and general purpose cameras.

Our TCNs capture low-, intermediate-, and high-level temporal (or spatiotemporal) cues from raw sensor signals, using a hierarchy of temporal convolutional filters. These learn low-level filters that are comparable to our earlier action primitives and high-level filters that capture transitions between actions. Not only do TCNs achieve superior performance, but in contrast to our other methods, in which inference was performed offline in batch, we demonstrate online TCNs which can be used for real-time applications. Lastly, we discuss parallels between work in the segmentation and action detection literatures, and show that TCNs can achieve state-of-the-art performance on both tasks.

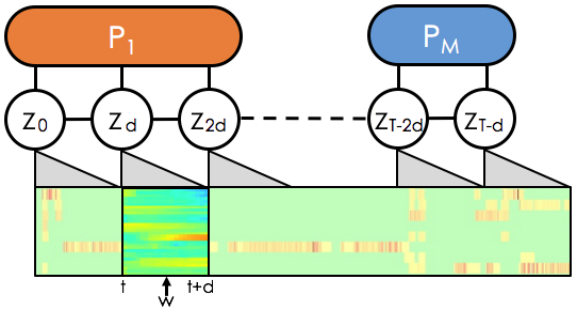
Our models achieve state-of-the-art performance on most of these datasets and in some cases close the gap between performance using domain-specific sensors and general video-based solutions. Furthermore, in Chapter 6 we address additional applications of the models developed throughout to the domain of surgical workflow analysis.

In Chapter 7, we address many of the practical implications and limitations of our models for real-world tasks. We highlight interesting future directions for fine-grained action modeling and describe potential pathways for enabling this work at scale.

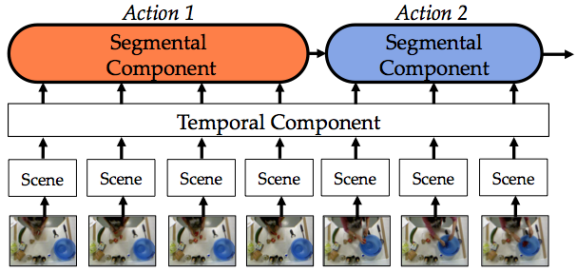
1.3 Thesis Statement

We posit that temporal convolutional filters can capture complex time-series patterns exhibited in sensor sequences, regardless of sensing modality, for use in fine-grained

Chapter 3: Latent Convolutional Time-series Model



Chapter 4: Segmental Spatiotemporal CNN



Chapter 5: Encoder-Decoder Temporal Convolutional Network

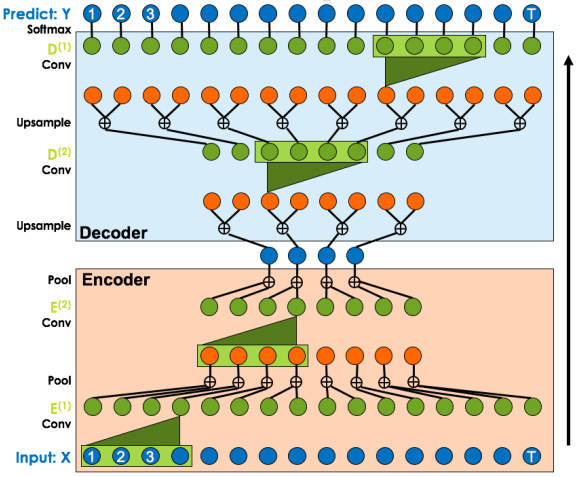


Figure 1.4: The resulting models described in each of the core chapters of this thesis.

action segmentation problems. We believe that when these filters are defined to operate over short intervals of time they will capture how sub-actions correlate with local changes in sensor values and when defined to operate over longer intervals of time they will capture properties including action duration and action transitions.

1.4 Contributions

The topics in this thesis can be categorized into video-based activity recognition, surgical data modeling, and robotics and have been written about in a series of five first-author conference papers (including one in-review), two workshop papers, and a series of collaborations. A breakdown of the chapters of this thesis and the corresponding papers in which they were described is as follows.

Chapter 3 is mostly based on

- *Learning Convolutional Action Primitives for Fine-grained Action Recognition*, **Colin Lea**, René Vidal, Greg Hager, ICRA, 2016.
- *An Improved Model for Segmentation and Recognition of Fine-grained Activities with Application to Surgical Training*, **Colin Lea**, Greg Hager, René Vidal, WACV, 2015.

Chapter 4 is based on

- *Segmental Spatiotemporal CNNs for Fine-grained Action Segmentation*, **Colin Lea**, Austin Reiter, René Vidal, Greg Hager, ECCV, 2016.

- *Sensor Substitution for Video-based Action Recognition*, Christian Rupprecht*, **Colin Lea***, Federico Tombari, Nassir Navab, Greg Hager, IROS, 2016.[2]

Chapter 5 is based on

- *Temporal Convolutional Networks for Action Segmentation and Detection*, **Colin Lea**, Michael D. Flynn, René Vidal, Austin Reiter, Greg Hager, arXiv, 2016. (in-review)

- *Temporal Convolutional Networks: A Unified Approach to Action Segmentation*, **Colin Lea**, René Vidal, Austin Reiter, Greg Hager, ECCV Workshop, 2016.

Chapter 6 uses the methods from the previous chapters and was written about in

- *Surgical Phase Recognition: from Instrumented ORs to Hospitals Around the World*, **Colin Lea**, Joon Hyuck Choi, Austin Reiter, Greg Hager, MICCAI: M2CAI Workshop, 2016. (Best workshop paper award)

The following collaborations and earlier work indirectly contributed to ideas in this thesis:

- *System Events: Readily Accessible Features for Surgical Phase Detection*, Anand Malpani, **Colin Lea**, Grace Chen, Greg Hager, IPCAI/IJCARS, 2016.

- *A Framework for End-User Instruction of a Robot Assistant for Manufacturing*, Kelleher Guerin, **Colin Lea**, Chris Paxton, Greg Hager, ICRA, 2015.

[2] * denotes equal contribution

- *Transition State Clustering: Unsupervised Surgical Trajectory Segmentation For Robot Learning*, Sanjay Krishnan, Animesh Garg, Sachin Patil, **Colin Lea**, Greg Hager, Pieter Abbeel, Ken Goldberg, ISRR, 2015.

- *3D Sensing Algorithms Towards Building an Intelligent Intensive Care Unit*, **Colin Lea**, James Fackler, Greg Hager, Russ Taylor and Suchi Saria, AMIA, 2013.

Chapter 2

Background and Related Work

In this chapter we discuss prior work from robotics, computer vision, and time-series modeling and introduce terminology which will be used throughout this thesis. It is important to understand the background on all three areas in order to appreciate the fundamental issues with current approaches to fine-grained action recognition.

2.1 Action Recognition Stratification

Within the activity recognition literature, terms including recognition, classification, and segmentation are often abused so we define a set terms that will be used consistently throughout this work.

Action Classification (trimmed): Given a video or sensor sequence that only consists of one action, classify that action.

Action Classification (untrimmed): Given a sequence of data that consists of one dominant action and some background class, classify the dominant action.

Action Localization: Given a sequence of data that consists of one dominant action and some background class, classify the dominant action and determine the starting and ending time.

Action Detection: Given a sequence of data with many actions, detect all instances of every action and the corresponding starting and stopping time for each. Typically there are "background" segments between actions which are not detected.

Temporal Action Segmentation: Given a sequence of data with many actions, densely label all time steps with an action class. This may include an explicit background class which must be detected.

Spatiotemporal Action Segmentation: Given a video sequence with many actions, densely label all pixels in each frame with the relevant action.

Action Recognition: This is an umbrella term for classification, segmentation, and the other aforementioned tasks, in which the input is some data sequence and the output is some information related to the presence, timings, or locations of actions.

Our primary focus is temporal action segmentation but in Chapter 5 we will draw parallels between action segmentation and action detection, and show that our models achieve state-of-the-art performance on both tasks.

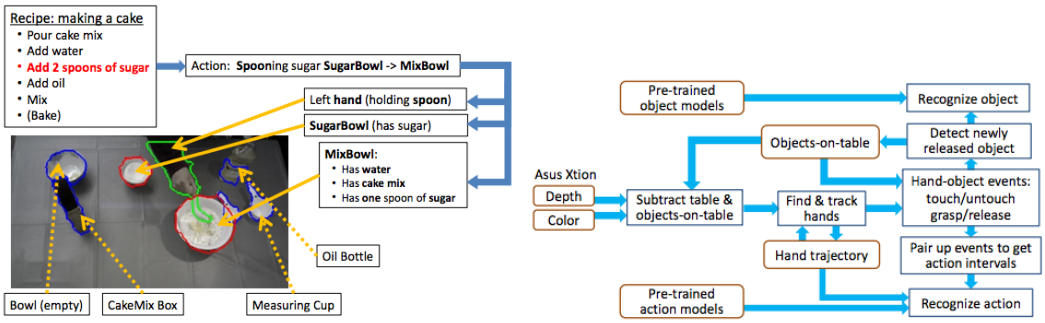


Figure 2.1: Images from [4] on modeling cooking activities (left) An example camera image and annotations (right) The pipeline used to recognize actions.

2.2 Activity Recognition in Robotics

Work in the robotics literature has explored action recognition in a diverse set of domains including manufacturing, robotic surgery, the home, and beyond. While conceptually some of the ideas may transfer across domains, given the large diversity, it difficult to assess the applicability of specific models between applications. As such, we describe prior work in the context of the domain that each paper was applied to.

2.2.1 Cooking Activities

Ramirez-Amaro *et al.* [22] segmented fine-grained actions in two cooking tasks – making pancakes and making a sandwich – in which the user's hands and the objects relevant to each task relevant could be detected reliably. They claimed that with accurate perception it is possible to segment actions with reasonable performance using simple predicate-based methods that do not require complicated temporal models. Their model used semantic reasoning, via a decision tree, with predicates like `hand_moving`

and `object_in_hand`. While their claim may be true in highly instrumented environments, contrary to our approaches, their methods likely would not perform well in most practical applications where objects cannot be reliably detected. Other papers in their line of work introduced more nuanced ontologies, described deployment in real-time settings, and described the application of their methods to other problem domains [26, 27, 28, 25].

Lei *et al.* [4] classified a set of actions within a cake-making activity using a set of hand-crafted features derived from object and hand trajectories. They used an RGBD camera and an object detector trained on objects relevant for the given task. Their features were used as input into a Hidden Markov Model to classify actions such as `chop`, `mix`, and `pour`. While their approach performed well, the solution is limited because they assume actions have already been segmented temporally. As we show later, if there is known temporal segmentation then results tend to be much higher than without.

Stein and McKenna [12] introduced the 50 Salads dataset for segmenting fine-grained actions in the context of making a salad. They applied standard frame-wise models, including Naive Bayes and Random Forests, with hand-crafted features derived from the location and usage of a set of ten objects. While their dataset is very good – we use it as a running dataset throughout our work – their performance is insufficient for practical applications. Furthermore, their methods require modifying kitchen tools with costly accelerometers for deployment. In this thesis, we start by

building models using accelerometer or other sensor data, but then generalize to video-only solutions. Stein and McKenna also look at the effect of using user-agnostic versus user-specific recognition models [29]. Our latent variable models are user-agnostic but are capable of modeling multiple stylistic variations of each action.

2.2.2 Surgical Activities

Work recognizing actions from surgical training tasks tends to use robot kinematic data such as the gripper pose and velocity from a daVinci robot [30, 31, 15]. Much of this work modeled surgical actions using variations on Hidden Markov Models [30, 31, 32] and Linear Chain Conditional Random Fields [33, 15]. These models classify each action using a linear combination of the sensor signals at each individual frame and the predicted action at the previous time interval. Varadarajan [31] modeled actions using a Switching Linear Dynamic System (S-LDS). His solution splits the signal into 15 frame partitions and fits an LDS to each. This is similar in spirit to our action primitives, which we describe in Chapter 3, but our approach uses much longer temporal windows and learns how the signals changes nonlinearly within each action. Tao *et al.* [33] used a Markov Semi-Markov model, which predicts a small set of action segments instead of individually predicting the action at each individual frame. We use this segmental idea within the context of our baseline model in Chapter 3.

Krishnan *et al.* [34] introduced a non-parametric Bayesian approach for clustering trajectories in surgical data using the JIGSAWS dataset. This model detects transi-

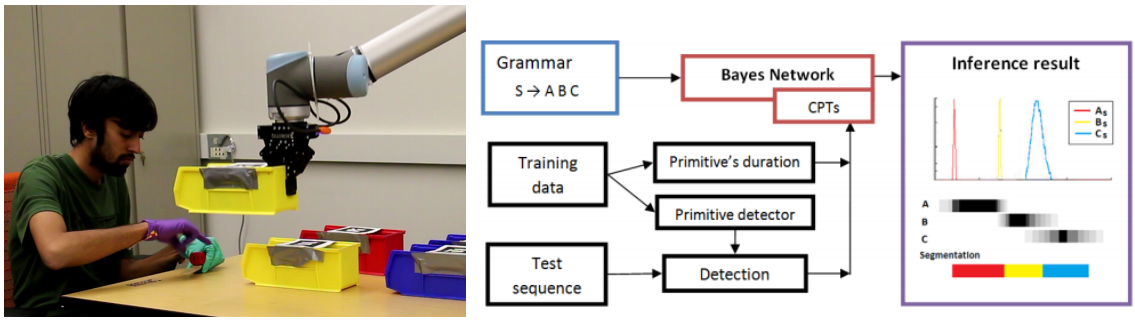


Figure 2.2: Images from [5] and [6] on modeling manufacturing activities (left) An example image of the user interacting with the robot (right) A schematic of the time-series model.

tions between actions based on changes in the linear dynamics. In their experiments, they showed that clusters tended to correspond to semantically meaningful partitions of the data, which were similar to the true labeled actions.

Zia *et al.* [35] classified surgical skill using a different approach, where instead of first recognizing actions, skill is computed directly from a set of features computed on the video. Their features were based on Discrete Fourier and Cosine Transforms applied to spatiotemporal interest points. They achieved high performance, however, their approach required the participants to wear special colored gloves to make the perception problem easier.

2.2.3 Human Robot Assembly Activities

Vo and Bobick [5] introduced the Sequential Interval Network (SIN) for action segmental in applications of human-robot interaction. SIN requires a known task model, as defined by a Context Free Grammar, to predict the start, stop, and type of each

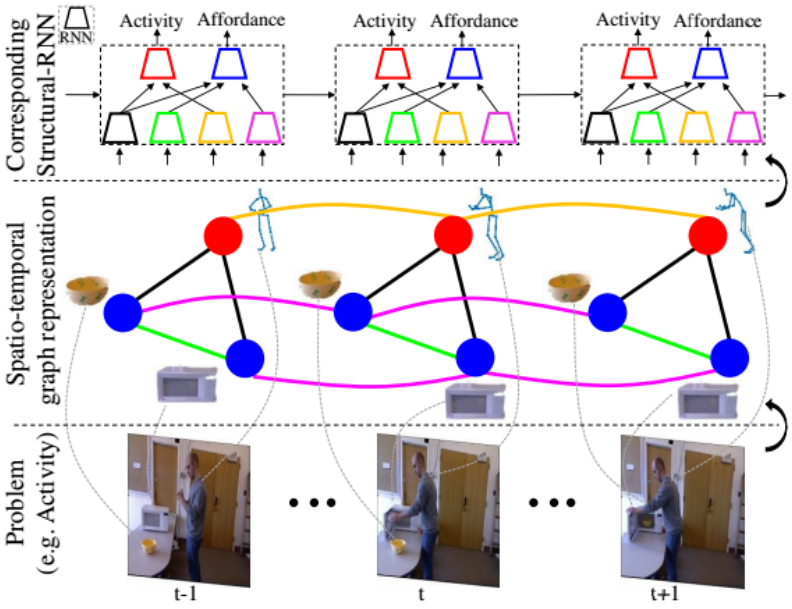


Figure 2.3: The Spatiotemporal RNN model proposed by Jain *et al.* [7] as applied to activities of daily living.

action segment. They applied their model to a toy assembly dataset in which a user alternates between grabbing parts of a toy airplane (e.g., wing, rudder) from a bin and adding it to an assembly. They detected the user's hand positions and the bin locations and used them as features in their model. Their Sequential-Interval Network (SIN) is similar to our segmental approach in Chapter 3, they rely on a fixed grammar to detect actions. Hawkins *et al.* [6] applied this model to a human robotic interaction task.

2.2.4 Daily Living Activities

Koppula and Saxena [36] proposed a Conditional Random Field model for activities of daily living that captures object-object and action-object relationships. This

model generates many action segment hypotheses by sampling and evaluating possible graph structures that encode these relationships. They evaluated this model on the Cornell Activity Dataset (CAD-120) which is composed of 120 RGB-D videos with ten activities, including making cereal or taking medicine, each performed by four users. Koppula *et al.* [37] extended this to the case of action prediction by sampling future states using an Anticipatory Temporal Conditional Random Field. More recently, Jain *et al.* [38] introduced a spatiotemporal Recurrent Neural Network with Long Short Term Memory (LSTM), that better captures the aforementioned object and action relationships over time than the CRF-based approach. This was applied to multiple datasets including CAD-120. They also showed impressive results predicting human motion by sampling future states from their RNN model. Jain used a simplifed version of this to autonomous driving to anticipate the actions of users [39].

Earlier work by De la Torre *et al.* [40] introduced the CMU Multimodal Activity Dataset (CMU-MMAC) which contains video and accelerometer data of people making food in an instrumented kitchen. They applied baseline methods, including Hidden Markov Models and Nearest Neighbors, to the sensor data and achieved a modest 38.4% accuracy [41]. More recently, Carvajal *et al.* [42] introduced a Fisher Vector-based approach using local temporal windows from video which improved results slightly to 40.9%.

2.2.5 Manipulation Activities

There has been recent interest in understanding the fundamentals of *manipulation actions*. These actions are categorized based on how a human interacts with an object such that the state of the object changes as a function of the applied action. For example, in a cutting task the transition from a whole cucumber to two halves is modeled as a change from one object segment to two. Yang *et al.* [43, 44] defined a set of object states and corresponding transitions to learn how objects change as a consequence of an action. While this work provided interesting insights into action-object relationships, it required a clean foreground-background separation. In the same line of work, Yang *et al.* modeled the relationship between manipulation actions and natural language [45], predicted how a user grasps an object [46], and learned how to ground perception with semantics [47].

2.3 Activity Recognition in Computer Vision

New spatiotemporal feature representations [9, 48] and massive datasets like ActivityNet [8] have catalyzed progress towards large-scale action recognition in recent years. In the large-scale case, the goal is to classify diverse actions like skiing and basketball, so it is often advantageous to capture contextual cues like the background appearance. In contrast, despite active development on fine-grained action recognition (e.g. [49, 50, 51, 52, 53]) progress has been comparatively modest. Many of these

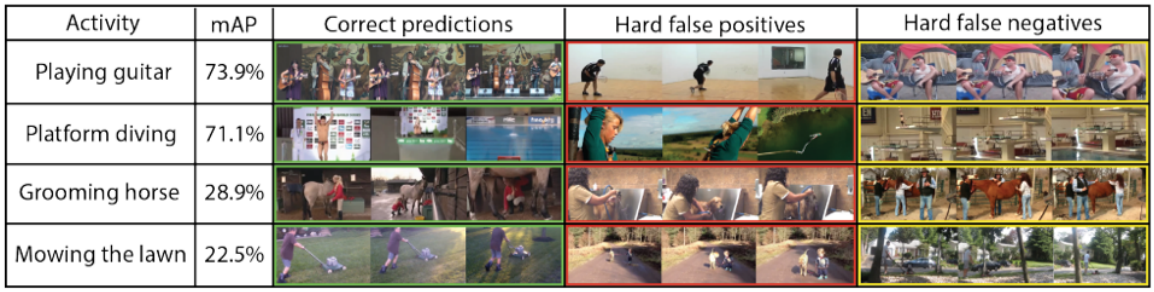


Figure 2.4: Evaluation from [8] on the large-scale ActivityNet dataset. Note the large diversity in the action types.

models do not capture the nuances necessary for recognizing fine-grained actions such as subtle changes in object location.

Research from the early 2000s in the computer vision community focused on small datasets, including KTH [54] and Weissman [55], with contrived activities like `walking` in a field versus `running` in a field. These datasets are considered solved, as the state-of-the-art accuracy in each is above 98%. Here, we describe holistic features, which are the dominant style of feature representation, discuss large-scale classification, which is the most common form of action recognition, and then describe progress on fine-grained action recognition.

2.3.1 Holistic Approaches

Holistic approaches typically refer to Bag of Words like models that ignore the spatial structure within an image or video clip. The use of spatio-temporal features with a bag of words representation is standard for large-scale [9, 56, 57, 7, 58, 59] and fine-grained [49, 50, 51, 57, 60] action analysis. The typical baseline represents a given

clip using Improved Dense Trajectories (IDT) [9] with a histogram of dictionary elements [49] or a Fisher Vector encoding [9]. Dense Trajectories concatenate HOG, HOF, and MBH texture descriptors along optical flow trajectories to characterize small spatio-temporal patches. Empirically they perform well on large-scale tasks, in part because of their ability to capture background detail (e.g. sport arena versus mountaintop). However, for fine-grained tasks the background is often constant so it is more important to model objects and their relationships. These properties are typically not captured by holistic approaches. Furthermore, the typical image patch size for IDT (neighborhood=32px, cell size=2px) is too small to extract high-level object information.

2.3.2 Convolutional Approaches

While recent work has extended CNN models to video [48, 7, 61, 62, 63, 64, 65], often results are only superior when concatenated with IDT features [7, 63, 64]. These models improve over holistic methods by encoding spatial and temporal relationships within an image. Several papers (e.g. [48, 61, 65]) have proposed models to fuse spatial and temporal techniques. While each achieve state of the art, their models are only marginally better than IDT baselines. Our approach in Chapter 4 is similar in that we propose spatiotemporal CNN, but our temporal filters are applied in 1D and are much longer (e.g. 10 seconds).

Despite success in classification, large-scale approaches are inadequate for tasks

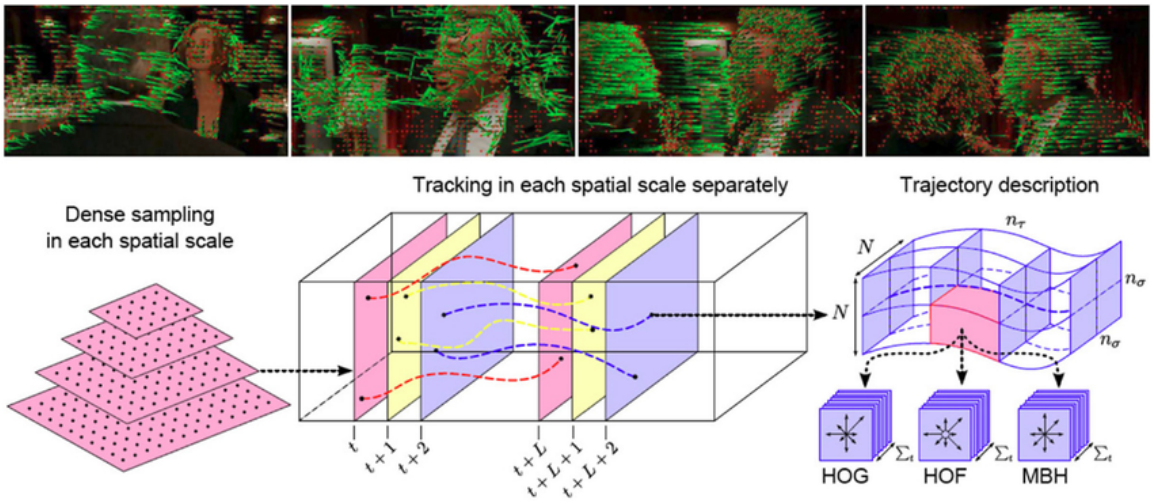


Figure 2.5: Improved Dense Trajectories [9] (top) Example visualizations (bottom) Pipeline for extracting features from video.

like action localization and detection which are more similar to our fine-grained task. In the 2015 THUMOS large-scale action recognition challenge[1] the top team fused IDT and CNN approaches to achieve 70% mAP on classification. However, the top method only achieves 18% (overlap ≥ 0.5) for localization. Heilbron *et al.* [8] found similar results on ActivityNet with 11.9% (overlap ≥ 0.2). This suggests that fundamental methodological changes are necessary for identifying and localizing actions regardless of fine-grained or large-scale.

Moving to fine-grained recognition, recent work has combined holistic methods with human pose or object detection. On MPII Cooking Rohrbach *et al.* [49] combine IDT with pose features to get a detection score of 34.5% compared to 29.5% without pose features. Cheron *et al.* [52] show that if temporal segmentation on MPII is known then CNN-based pose features achieve 71.4% mAP. While this performance is

[1]THUMOS Challenge: `http://www.thumos.info/`

comparatively high, classification is a much easier problem than detection. Object-centric methods (e.g. [50, 51, 53]), first detect the identity and location of objects in an image. Ni *et al.* [53] achieve 54.3% mAP on MPII Cooking and 79% on the ICPR 2012 Kitchen Scene Context-based Gesture Recognition dataset. While performance is state of the art, their method requires learning object models from a large number of manual annotations. In our work we learn a latent object representation without object annotations. Lastly, on Georgia Tech Egocentric Activities Li *et al.* [51] use object, egocentric, and hand features to achieve 66.8% accuracy for segmentation and classification versus an IDT baseline of 39.8%. Their features are similar to IDT but they use a recent hand-detection method to find the most important locations in each image. In contrast to our approach, their method does not appear to capture the relationship between objects.

2.3.3 Time-series Models

Some of the models that we build upon are derived from those which originally came from the machine learning and statistics literatures but are frequently used in computer vision. In this section we briefly describe work on time-series models including Hidden Markov Models, Conditional Random Fields, segmental models, and Recurrent Neural Networks.

Conditional Random Fields (CRFs) are a family of discriminate graphical models that are used to predict a set of labels (e.g., actions) given data. In our work we

consider the case where we have a stream of input data over time and we have a set of labels, one per time-step. CRFs are designed to encode the relationships between the data and labels across time. For an introduction to these models from a machine learning perspective see [66] or from a computer vision perspective see [67]. CRFs have become common for action segmentation and classification (e.g., [33, 68, 69]) and often capture multiple types of interactions between labels. For example, a unary term may model the relationship between the data at the current time step with the label at the current time step and a pairwise term may capture the likelihood of transitioning between two actions at two sequential time steps.

One way of framing the problem of action segmentation is that we want to extract a small but variable number of actions from a longer sequence, so some work has focused on the development of segmental models. The key idea in these models is that instead of having a unary term defined for each individual frame, the unary is defined across a variable-duration segment. The earliest work in this area dates back to the 1980s by Ferguson [70], but much of the recent work is based on the more recent Semi-Markov CRF by Sarawagi and Cohen [71] who approached the problem of Named Entity Recognition problem in Natural Language Processing. Between statistics, machine learning, and computer vision there have been hundreds of papers on semi-Markov Models [72]. While most of this work has been in the speech processing community, recently these models have been used for activity recognition in the computer vision and sensing communities [57, 68, 73, 74, 33, 75, 69]. For exam-

ple, Shi *et al.* [75] proposed a discriminative Semi-Markov model with a histogram of cuboid and shape context features to recognize actions in the KTH and CMU MoBo datasets. They also evaluate their model on a contrived activity dataset that they generated which included actions such as "walking," "bending," and "drawing." Pirsiavash and Ramanan [57] introduced a similar model but in the form of a segmental regular grammar. They defined their segmental unary function using normalized histograms using Dense Trajectories and other local video features. In many of the recent segmental action segmentation solutions, segments are simply modeled using histograms of holistic features. In our approach we use a segmental model in tandem with learned temporal or spatiotemporal features.

Many of the aforementioned models were influenced by earlier work on Hidden Markov Models, which date back to the 1970s and 1980s (e.g., [76, 77, 78]). The vanilla HMM model is comprised of two terms: one that captures a set of hidden states, which in our context may correspond to subactions, and one that captures the transitions between subactions. Much of this progress came from the speech processing community and was used for decades in Automated Speech Recognition (ASR) systems, as described by the recent ARS overview by Yu and Deng *et al.* [79]. The thesis of Kevin Murphy [80] describes many HMM variants, and generalizes some of the earlier models. Some examples include auto-regressive HMMs, which model the probability of generating the data at a given time step using both the latent states and the data at previous time steps; hierarchical HMMs, which models several layers

of latent states (e.g., one layer represents sub-actions, one represents actions, and one represents sequences of actions); and variable-duration HMMs, which employ a semi-Markov assumption using a new term that captures the temporal duration of each class. The thesis of Murphy influenced much of the other work on segmental CRFs and HMMs which we describe later.

It is worth noting that much of the work on HMMs has been employed for action classification. Separate HMMs are typically trained on each action class, meaning, there may be one HMM trained on a "mixing action and another on "cutting. Each state of an HMM may correspond to a subaction of the given class, thus, additional processing needs to be performed to segment each of the actions across time. Note, at the same time as many of these HMM models were being developed, there was also work being done on Linear Dynamical Systems (LDS) (e.g., [81, 82, 83, 84]), a continuous analog of HMMs. One advantage of an LDS is that their are more theoretically sound metrics for comparing LDSs, and determining which class of LDS best fits a segment of data, compared to HMMs [84].

Recently, there has been significant interest in Recurrent Neural Networks (RNNs), specifically those using Long Short Term Memory (LSTM) (e.g. [65, 85, 86, 87]). RNNs and Latent Linear Chain CRFs are conceptually similar, and in Chapter 3 we compare them in more detail. Simply put, in RNNs, there is a temporal component that captures transitions across learned latent states, whereas in CRFs, there is a temporal component that captures transitions in high-level action labels. While the

performance of RNNs with LSTM is often impressive, they are hard to interpret due to their complex series of gating functions. In contrast to the models in this thesis, we explicitly learn how latent states transition in a way that is easier to interpret and visualize. Ours is more similar to those in speech recognition (e.g. [88, 89, 85]) which learn phonemes using 1D convolutional filters.

2.4 Datasets

Historically, most action recognition datasets from the computer vision community were developed for classifying individual actions using pre-trimmed clips. There have been recent datasets for fine-grained detection and segmentation, however they often contain too few users or an insufficient amount of data to learn complex models. MPII Cooking [49] has a larger number of videos but some actions are rarely performed. Specifically, seven actions are performed fewer than ten times each. Action segmentation datasets from the robotics community are also limited in scope or amount of data. For example, Ramirez-Amaro *et al.* [26] introduced the TUM Cooking dataset which includes two cooking activities. The pancake activity is only performed by one user and the sandwich activity has five users. There are many RGBD-based gesture recognition datasets, which typically offer RGB, depth, and skeleton data, but are defined for action classification instead of segmentation. See the recent survey by Zhang *et al.* [90] for RGBD-based action recognition datasets.

In Chapters 3 and 4 we assess our models on 50 Salads and JIGSAWS datasets

which both contain multi-modal data and many instances of each action. In Chapters 5 and 6 we assess these models on several additional the computer vision, surgical training, and surgical workflow communities. We discuss 50 Salads and JIGSAWS in depth in the next subsection and briefly highlight some of the computer vision datasets below.

2.4.1 University of Dundee 50 Salads

The University of Dundee 50 Salads [12] dataset consists of time-synchronized video, depth, and accelerometer data. Twenty-five users each make a salad in two different videos for a total of 50 trials. Each trial is 5-10 minutes long. As shown in Figure 4.1, a static RGBD camera is mounted faced down pointed at the user preparing a salad. The motion of each kitchen tool is captured via an accelerometer embedded in the handle. This data can be used to indicate which tools are in use at any given time. In total there are 10 accelerometers which are located on the `plate`, `pepper dispenser`, `bowl`, `oil bottle`, `large spoon`, `dressing glass`, `knife`, `peeler`, `small spoon` and `chopping board`.

This dataset contains four action label granularities. At the highest level there are three action classes: `cut_and_mix_ingredients`, `prepare_dressing`, and `serve_salad`. There are 17 mid-level actions such as `add_vinegar`, `cut_tomato`, `mix_dressing`, `peel_cucumber`, `place_cheese_into_bowl`, and `serve_salad`. The third granularity splits each mid-level action into three sub-actions: `start`, `core`, and `finish`, for a

total of 51 actions. Following the work of [12], we also evaluate using a second mid-level granularity that consists of 10 actions that can reasonably be recognized using the sensor-laden tools. Note that in [12] this was called "eval." Here we use the term "higher-level," All label sets also include a background class used when no action is occurring. Tables 2.1 and 2.2 show the actions for each granularity.

50 Salads (Highest-level)
background
cut and mix ingredients
prepare dressing
serve salad

50 Salads (Higher-level)
background
add dressing
add oil
add pepper
cut
mix dressing
mix ingredients
peel cucumber
place
serve salad onto plate

50 Salads (Mid-level)
background
add dressing
add oil
add pepper
add salt
add vinegar
cut cheese
cut cucumber
cut lettuce
cut tomato
mix dressing
mix ingredients
peel cucumber
place cheese in bowl
place cucumber in bowl
place lettuce in bowl
place tomato in bowl
serve salad onto plate

Table 2.1: Mid-, higher-, and highest-level action labels for the 50 Salads datasets

Figure 2.6 depicts the ground truth labels for all 50 sequences in 50 Salads. Note that there is large variability in how each user makes a salad. Some action instances are much longer in duration and the order of the actions varies substantially. Both of these should be apparent in the visualization.

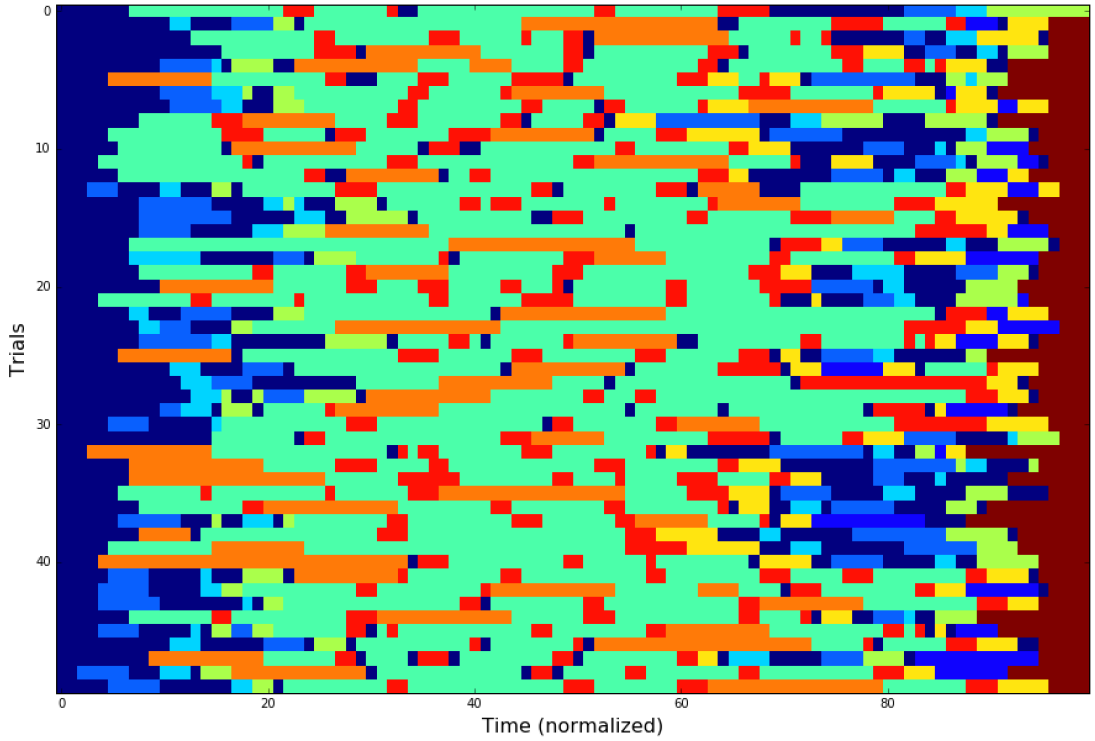


Figure 2.6: Action labels for each sequence in 50 Salads using the higher-level granularity. Each row corresponds to one video in the dataset and each color corresponds to a different action label. The colors range from blue to red as indicated in Table 2.1. For visualization purposes, all videos are normalized to the same total duration. Note the large variation in properties like action duration and the ordering of actions.

50 Salads (Low-level)		
add dressing(pre)	add dressing(core)	add dressing(post)
add oil(pre)	add oil(core)	add oil(post)
add pepper(pre)	add pepper(core)	add pepper(post)
add salt(pre)	add salt(core)	add salt(post)
add vinegar(pre)	add vinegar(core)	add vinegar(post)
cut cheese(pre)	cut cheese(core)	cut cheese(post)
cut cucumber(pre)	cut cucumber(core)	cut cucumber(post)
cut lettuce(pre)	cut lettuce(core)	cut lettuce(post)
cut tomato(pre)	cut tomato(core)	cut tomato(post)
mix dressing(pre)	mix dressing(core)	mix dressing(post)
mix ingredients(pre)	mix ingredients(core)	mix ingredients(post)
peel cucumber(pre)	peel cucumber(core)	peel cucumber(post)
place cheese in bowl(pre)	place cheese in bowl(core)	place cheese in bowl(post)
place cucumber in bowl(pre)	place cucumber in bowl(core)	place cucumber in bowl(post)
place lettuce in bowl(pre)	place lettuce in bowl(core)	place lettuce in bowl(post)
place tomato in bowl(pre)	place tomato in bowl(core)	place tomato in bowl(post)
serve salad onto plate(pre)	serve salad onto plate(core)	serve salad onto plate(post)
background		

Table 2.2: Low-level action labels for the 50 Salads datasets

2.4.2 JIGSAWS

The JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) [91] has three common tasks for robotic surgery training – suturing, needle passing, and knot tying – each of which are performed on bench top phantoms. Videos and robot kinematics were collected from a daVinci surgical robot, and an example of the suturing task is shown in Figure 3.1. These activities are each decomposed into about 10 unique action labels such as `insert_needle_into_skin`, `tye_a_knot`, `transfer_needle`, and `drop_needle_at_finish`. The full list is found in Table 2.3. Each task has between 26 and 39 trials performed by up to 8 users. Videos are around two minutes long and contain 15 to 20 actions per video.

Figure 2.7 depicts the ground truth action labels for each sequence. Note that

JIGSAWS
Reach for needle
Position needle at insertion point
Push needle through tissue
Transfer needle between tools
Move needle to center
Pull thread with left
Orient needle
Tighten suture with right
Loosen suture
Drop suture at finish

Table 2.3: Action labels for suturing in the JIGSAWS dataset.

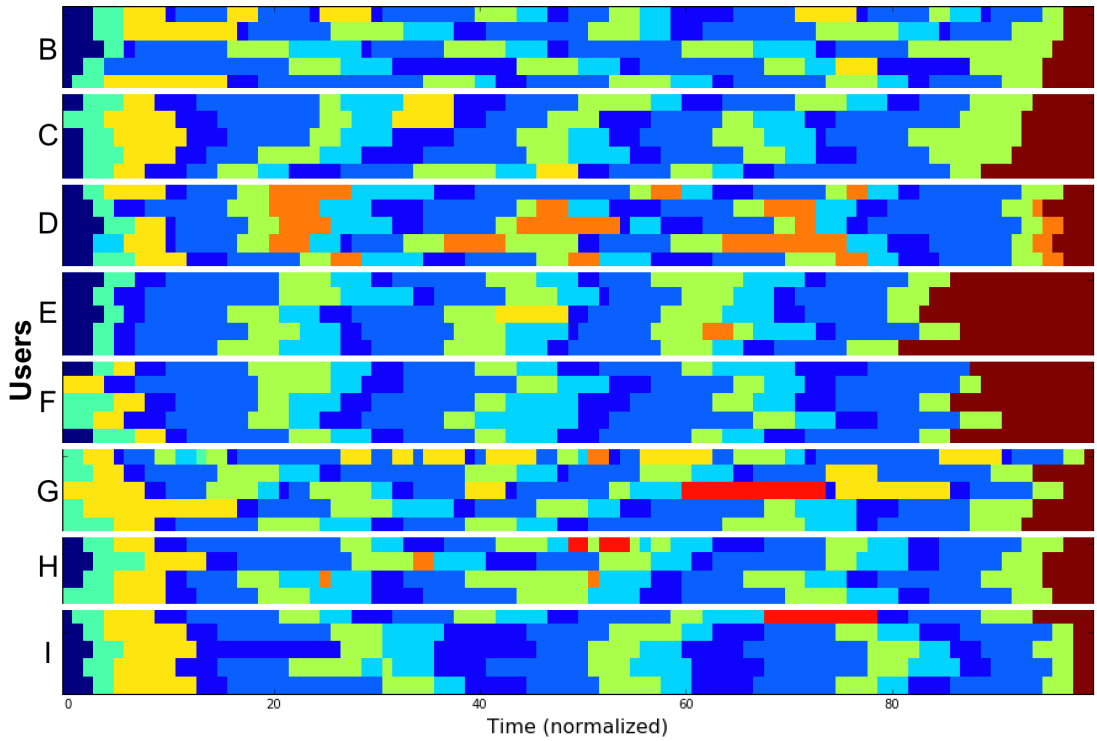


Figure 2.7: Action labels for each sequence in JIGSAWS. Each row corresponds to one video in the dataset and each color corresponds to a different action label. Recall that each user performs the suturing activity 5 times. The colors range from blue to red as indicated in Table 2.3. For visualization purposes, all videos are normalized to the same total duration. Note the large variation in properties like action duration and the ordering of actions and that some actions are only performed by a small number of users.

there is large variation in the duration of each action, the ordering of actions, and the number of actions per sequence. Later we will find that the actions for users D, G, and H are the hardest to predict regardless of which model is used. It is clear from the visualizations that there are larger discrepancies in the temporal action patterns for these users.

While the JIGSAWS dataset contains 76 kinematics features from a da Vinci robot, including both surgeon- and patient-side positions, rotations, and velocities, we find that this is highly redundant. Furthermore, we want to ensure our models are applicable to a wide range of surgical robots for which we might only have position and velocity information. For all of our sensor experiments on JIGSAWS, we use the patient-side robot positions, velocities, and end effector gripper angles for the left and right tools.

2.4.3 Additional Datasets

Georgia Tech Egocentric Activities [23] has 28 videos across seven tasks and most work has used a very fine-grained set of 61 actions such as `pour mayonnaise on bread with cheese`. Many of the actions are only performed a few times, and there is large variability between each of those times, which makes it hard for our models to capture characteristics (e.g., object usage, motion patterns) that are indicative of those actions. We use the smaller set of nine action classes used by Singh *et al.* [92] which contains more instances per action. The environment and actions are fairly

contrived. For example, the activity "making a hotdog", consists of taking a piece of white bread, putting a raw hot dog on it, and adding condiments. Note that the CMU Kitchen dataset [93] is in the same domain and contains much more data. It has video and sensor data for 18 users who make five recipes. Unfortunately, researchers typically only use videos for seven users making "brownies" recipe [93, 92]. Thus it is unclear how results on this little amount of data generalize to other scenarios.

The MERL Shopping dataset [24] was recently developed for action detection in surveillance applications. It consists of 106 videos in which a shopper walks along a (manufactured) grocery store aisle and manipulates objects on the shelves. There are five actions plus a background class: `reach to shelf`, `retract hand from shelf`, `hand in shelf`, `inspect product`, `inspect shelf`. In total there are 41 unique shoppers. The data is captured from an overhead camera which is in the same position with respect to the environment in all videos. Each action is typically only a few seconds in duration.

There are two other datasets used by recent work which we will not describe throughout the dissertation, but which are worth noting. These are the MPII Cooking dataset and the Kitchen Scene Context based Gesture Recognition dataset (KSCGR).

In MPII Cooking, users follow a complex recipe, such as making a cake, and the task is to classify or segment very fine-grained actions. In total there are 65 actions. Unfortunately, some of these actions, like "smelling," very rarely occur. Seven actions are performed fewer than ten times. It is difficult to build a model that can effectively

distinguish these from other actions given the large amount of variability between each instance. In addition, actions are sparse, meaning there is significant use of a background class in most videos. KSCGR was introduced at ICPR 2012 and contains overhead videos of five users making five different recipes on a stove. It includes eight actions such as `mixing`, `baking`, and `cutting`. It In contrast to other datasets, multiple annotators labeled each video and the results were averaged. Therefore, it is reasonable to assume the labels are of higher quality.

Chapter 3

Conditional Random Field-based Time-series Models

In this chapter we introduce a baseline Conditional Random Field-based time-series model and corresponding temporal feature representation, that will be used to predict a sequence of actions given sensor data such as robot kinematics or accelerometer values.

3.1 Introduction

In many robotics applications it is common to instrument an environment with domain-specific sensors such as accelerometers, encoders, or force sensors. When used for activity analysis, these sensors are typically installed in a way that makes it

easy to distinguish what action is happening at a given time directly from the sensor data. This was emphasized by Ramirez-Amaro *et al.* [22] who showed that in a constrained environment – when the locations of salient objects can be easily estimated – primitive actions like *object_in_motion* are very to easy detect. While we would like to be able to recognize actions solely from video footage, video-based analysis is more difficult because it may require identifying objects in an image and modeling the relationships between these objects as a function of the actions taking place. Therefore, we start by addressing the problem of action recognition using domain-specific sensors and then in the next chapter extend this to video.

In this chapter, we focus on two domains: robotic surgery training and cooking. For robotic surgery, we assume access to position and velocity-based sensors for each of the robot end effectors from a daVinci surgical robot. For cooking, we assume that all kitchen utensils are instrumented with accelerometers, as in 50 Salads. While it may not be realistic to instrument an everyday kitchen with accelerometers, later in this thesis we describe how we can leverage these sensor-based results to develop a better video-based solution.

It is important to understand some of the underlying challenges in modeling actions from sensor data before starting to define a model. Figure 3.1 shows example images from each domain and corresponding visualizations of the sensor signals where the color indicates the signal value. While it is not very common to visualize sensor data using 2D images, we find that it is very useful in identifying and visualizing

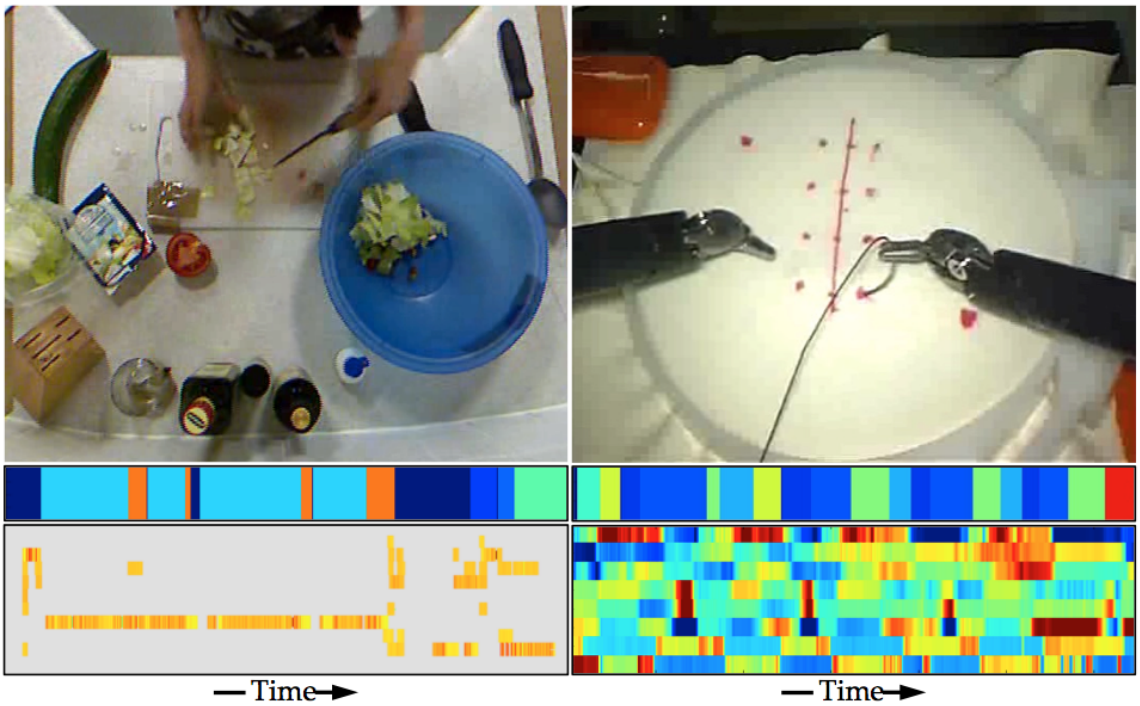


Figure 3.1: Sample data from University of Dundee 50 Salads and the JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS). The middle shows a corresponding action sequence where each color denotes an action label. The bottom depicts sensor signals where each row is an accelerometer or robot pose value over time.

action patterns.

We would like to capture two types of variability: local and global. In this context, local variability refers to changes in the sensor data within a given action segment. In Figure 3.2 (top) we see that the action "add dressing" is composed of three (unlabeled) motion patterns "use glass container," "use pepper shaker", and "use oil bottle." Our model should be able to capture these temporal patterns, each of which may last on the order of several seconds long. Furthermore, it is important to note that users may perform this action in very different manners. One person may start the action

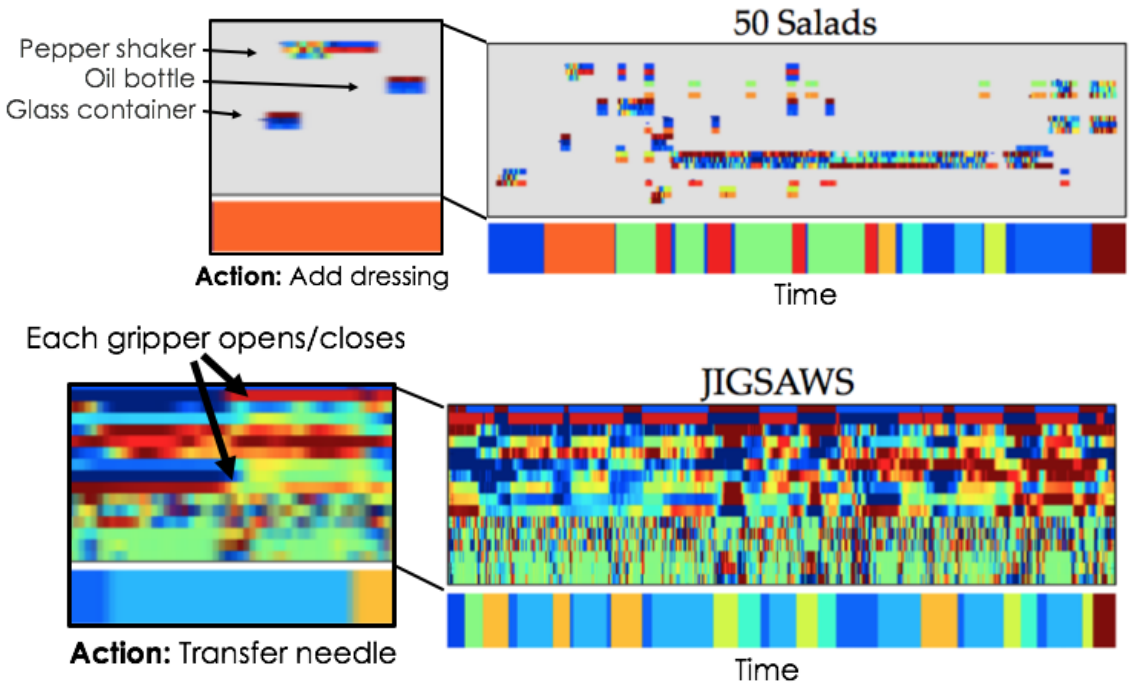


Figure 3.2: Sample data from (top) University of Dundee 50 Salads and (bottom) the JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS). In 50 Salads each row indicates the X, Y, or Z acceleration of a given object. In JIGSAWS the rows indicate sensor values corresponding to the gripper states (is the tool opened or closed), the positions of the left and right grippers, and their respective velocities.

using the glass container, whereas another user may start using the pepper shaker. We refer to the different ways in which users perform the same action as the user's "style." The same kinds of patterns can be seen in the JIGSAWS visualization in Figure 3.2 (bottom). In this case there is a clear change in gripper state when the user transfers the needle from one tool to the other.

Global variability refers to longer-range temporal patterns such as the ordering of actions. For example, in the suturing example –absent of any mistakes – the user should repeat the actions "position need," "push needle through tissue," "tighten suture," etc. four times in a row. Furthermore, certain actions, like "dropping the

suture at the finish" may only occur at the end of a trial. In Chapter 2 we showed example timelines for each of the datasets (e.g., Figure 2.7), which depicts many variations on how a user may perform the same task.

Our goal is to develop an improved approach for action segmentation that is agnostic to the specific sensors being used; the same methods should be applicable to accelerometer or robot kinematics data. We start by improving upon recent Conditional Random Field-based models (e.g., [33]), which have been effective on the JIGSAWS dataset. These models typically have a data component that captures local variability and a temporal component that captures global variability. We start by identifying two important limitations of common temporal components when applied to time series data. First, there is a large disconnect between the sampling rate of the data (e.g., 30 frames per second) versus the rate at which users perform actions (e.g., 1 action per 10 seconds). In linear chain models (e.g., Linear Chain CRFs), the temporal component only has a small impact on performance due to this disconnect. Second, we note that the duration of each action varies significantly, even within the same action class. For example, the "mixing ingredients" action in 50 Salads may last as short as 13 seconds or as long as a minute. In segmental models that incorporate a duration model (e.g., Semi-Markov CRFs), this again results in the temporal component only having a small impact on performance. We overcome these limitations by using a series of "skip chains," which capture long-range transitions between distant time-steps (instead of between sequential frames), and exploring a K-segment

inference algorithm that bounds the maximum number of segment predictions given the data component and skip chain temporal term (instead of bounding the duration of each segment).

To capture local variability across sensor modalities we use a learned temporal feature representation. These primitives capture how raw sensor signals, like robot position or object acceleration, change over time. In contrast with many approaches in robotics, which use hand-crafted features defined specifically for a given task, our approach is applicable to any sensor input. For each action class, we learn a set of temporal filters which are convolved with the input data. These are motivated by recent work with Convolutional Neural Networks where hierarchies of convolutional filters are learned for tasks like object classification (e.g., [94, 95]). One benefit of learning these filters in a CRF-based framework is that their weights are relatively interpretable. We find that the learned filters highlight common patterns found in the data, such as picking up or setting down an object in 50 Salads.

We incorporate these ideas into a latent CRF-based approach inspired by Tao *et al.* [33]. The latent aspect captures subactions and variations on how users perform each action. From the example above, these may capture the fact that a user may first use the glass container or the pepper shaker in the "add dressing" action. We jointly learn the parameters of each component of our model using a max-margin approach with a Latent Structural Support Vector Machine [96].

The final contribution of this chapter pertains to evaluation metrics. We find

that the metrics typically used for action segmentation are insufficient at capturing important issues for many robotics applications. Specifically, papers use frame-wise metrics like accuracy or precision/recall (e.g., [50, 12, 97]). Each of these metrics assumes that each frame is independent, which means that a method with many spurious over-segmentation errors may still achieve high accuracy. Over-segmentation errors in particular may preclude the user of a model in many applications, including surgical skill evaluation, especially in cases there the predicted ordering of actions is important. We propose the use of a segmental metric to address this limitation.

In summary, we describe four contributions in this chapter. First, we describe a CRF that overcomes limitations with common temporal models. Second, we model local variability within segments using a set of temporal convolutional filters. Third, we propose a constrained approach for segmental inference in these models. Lastly, we identify limitations in the current metrics for action segmentation and suggest alternatives. At the time of original publication, our models achieved state-of-the-art performance in both surgical and cooking domains.

As an aside, there has been a recent resurgence in the use of Recurrent Neural Networks (RNNs) for time-series modeling. In the appendix of this chapter (Section 3.8), we describe the differences between RNNs and CRFs and discuss how our skip chains and action primitives may be used to improve performance of RNNs.

3.2 Skip Chain Models

There has been a long history of developments in time-series models using linear chain and segmental models. Some recent examples include n-th order CRFs [98], Semi-Markov CRFs [71, 74, 75], and RNNs [99, 100, 101]. These models capture longer-range dependencies than traditional linear chain models, however, the computational complexity of their inference algorithms can become prohibitively expensive when actions are long in duration. Furthermore we show later that their performance – as measured by metrics such as accuracy – is not always superior to other linear chain models. In this section, we introduce the notion of a skip chain, which is an alternate method for capturing long-range temporal information which performs well and can be computed efficiently.

We first introduce the model, inference, and learning for the Skip Chain CRF and then generalize it to the latent variable case. Lastly, we introduce a segmental variant and corresponding inference algorithm that tends to reduce the number of over-segmentation errors, which are common with traditional semi-Markov approaches.

3.2.1 Skip Chain CRF

Let $X_t \in \mathbf{R}^F$ be a vector of F features (e.g. positions, velocities) at time t for $t \in \{1, \ldots, T\}$ and $y_t \in \{1, \ldots, C\}$ be the corresponding action (e.g. cutting, peeling) with class count C. We model the conditional distribution of the sequence of labels

$y = \{y_t\}_{t=1}^T$ given the sequence of features $X = \{X_t\}_{t=1}^T$ using a CRF model with Gibbs distribution $P(y|X) \propto \exp(E(X, y))$, where the scoring function

$$E(X, y) = \sum_{t=1}^{T} \phi(X, y, t) + \psi(y, t) + \pi(y, t) \tag{3.1}$$

gives the score of assigning labeling y to sequence X. Here, ϕ, ψ, and π denote the unary (data) scores, pairwise skip scores, and temporal priors respectively, all of which are described in detail in the following subsections. Each term will be a linear function of a set of weights and a set of features. See Figure 3.3 for a depiction of this model. d, as described later, is a parameter of the pairwise term that refers to the skip length.

Frame-wise Unary Term

Many time series models, including HMMs and most time-series CRFs, use a frame-wise action representation. This means that the unary score for each frame is a function of the data solely at that frame, which is independent of the data at surrounding frames. This score is usually a linear combination of weights $W \in \mathbf{R}^{C \times F}$ and biases $b \in \mathbf{R}^C$ and the data per time-step X_t. We index the row of weights for the y-th class as W_y. As such, the unary term is typically defined as

$$\phi(X, y, t) = W_{y_t} X_t + b_{y_t}. \tag{3.2}$$

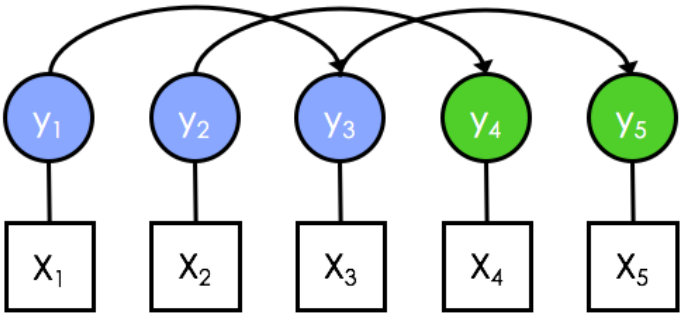


Figure 3.3: Skip Chain Conditional Random Field with a Skip Length parameter $d = 2$.

Later this will be replaced by our action primitive model.

Pairwise Skip Chain Term

Linear chain models like HMMs and CRFs are effective at modeling sequential data when the state (e.g. action) is changing with a high frequency. For example, in natural language processing tasks, like part-of-speech tagging, each consecutive word may correspond to a different label like `noun` versus `verb`. However, in our applications there is a disconnect between the frequency of data (e.g. 30 frames per second) and the frequency of actions (e.g. 0.2 actions per second). In this case, the Markov term has a very high self-transition probability. In Figure 3.4 we display pairwise transition matrices corresponding to three temporal models on JIGSAWS and 50 Salads. On the left we show the Markov case, which corresponds to the probability of transitioning between class a and class b at subsequent time steps (from $t-1$ to t). On the right we show the segmental case, described in Section 3.3, which corresponds to the probability of transitioning between class a and class b between subsequent segments.

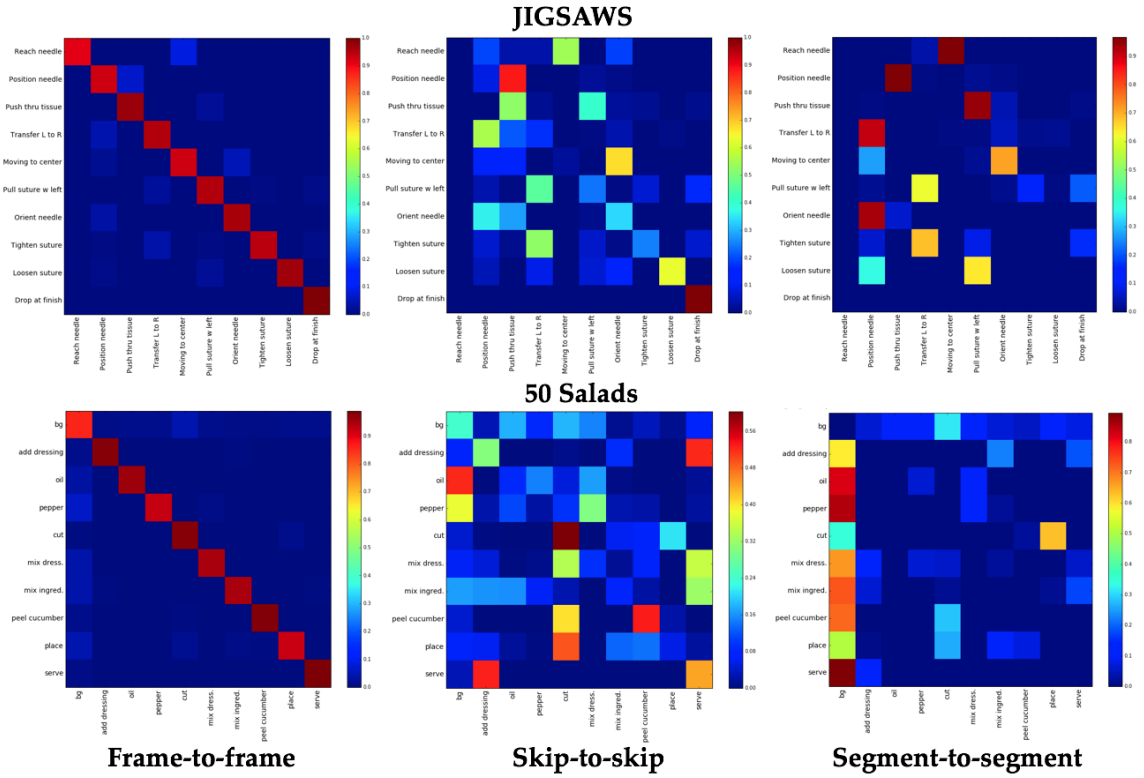


Figure 3.4: Pairwise transition matrices on the (top) JIGSAWS and (bottom) 50 Salads datasets. For both datasets: (left) The probability of transitioning between classes from $t-1$ and t. (center) The probability of transitioning between classes from $t-d$ and t. (right) The probability of transitioning between classes from segment to segment. For JIGSAWS $d = 100$ (3.33 seconds) and 50 Salads $d = 300$ (10 seconds). Red indicates high values and blue indicates low values. Notice that the matrices on the left have very high values along the diagonal (self-transitions) and very low values everywhere else.

In the center we show the probability of transitioning between actions using the skip chains described in this section. Note in the Markov case there is a very high rate of self-transitions, meaning between most consecutive frames the class label remains the same. In the segmental case, there is an issue in 50 Salads where most actions lead to the background class, and the background class may lead to any (non-background) class.

A pairwise skip chain term is a generalization of the Markov class transition term commonly used in linear chain models. These skip chains capture class transitions from time $t-d$ to t for a fixed skip length d. Intuitively, the probability of an action changing from class a to class b between these time steps is much higher than from $t-1$ to t, such that:

$$P(y_t = b | y_{t-d} = a) \gg P(y_t = b | y_{t-1} = a). \tag{3.3}$$

Empirically, using a delay d has a substantial effect on accuracy and better captures higher-order class transitions.

Note that in Natural Language Processing skip chains are used to capture relationships between related words within a sentence [102]. In our case, we are using skips to capture relationships between actions across long periods of time.

We model skip connections using a pairwise transition matrix, $U \in \mathbf{R}^{C \times C}$, indexed by the current label y_t and label y_{t-d}, which is d steps prior. The score corresponding to the pairwise transition model is

$$\psi(y, t) = \begin{cases} U_{y_{t-d}, y_t} & \text{if } t > d \\ 0 & \text{otherwise} \end{cases}. \tag{3.4}$$

We use a max-margin approach, as we describe later, in which $U_{a,b}$ is a score indicating whether class b should come d steps after class a. This term will be learned jointly

with our other terms in our scoring function to maximize performance. Note that in a probabilistic approach matrix U may be minus the log probabilities estimated for transitioning from class a to class b over an interval of d frames.

Temporal Prior Term

In many fine-grained applications, there may be a subset of actions that always happen at the beginning or end of a sequence. Similarly, there may be some actions that tend to happen earlier or later in the sequence. We use a temporal prior that captures the likelihood of assigning a class to any given time step. We do this by splitting the sequences into a set of intervals (e.g., the first $\frac{1}{10}$-th of the video, the second $\frac{1}{10}$-th, ...) and capturing which action tends to occur in that interval.

Figure 3.5 shows the probability that each action occurs at each (normalized) time step in the JIGSAWS and 50 Salads datasets. It is evident that certain actions tend to happen more often at the beginning or end of the sequence, but that there are oscillations between the other actions in the middle of a sequence.

It is important to note that the number of time-steps in a sequence, T, varies substantially between videos. The duration of one video may be twice and long as another. In order to model our temporal prior, we define T_o to be a canonical number of time steps in any sequence. Then, for any intermediate time step t, we define a canonical time step, $\tau_t \in \{1, T_o\}$, which is normalized such that $\tau_t = \lceil T_o * \frac{t}{T} \rceil$, where $\lceil \cdot \rceil$ is the ceiling function.

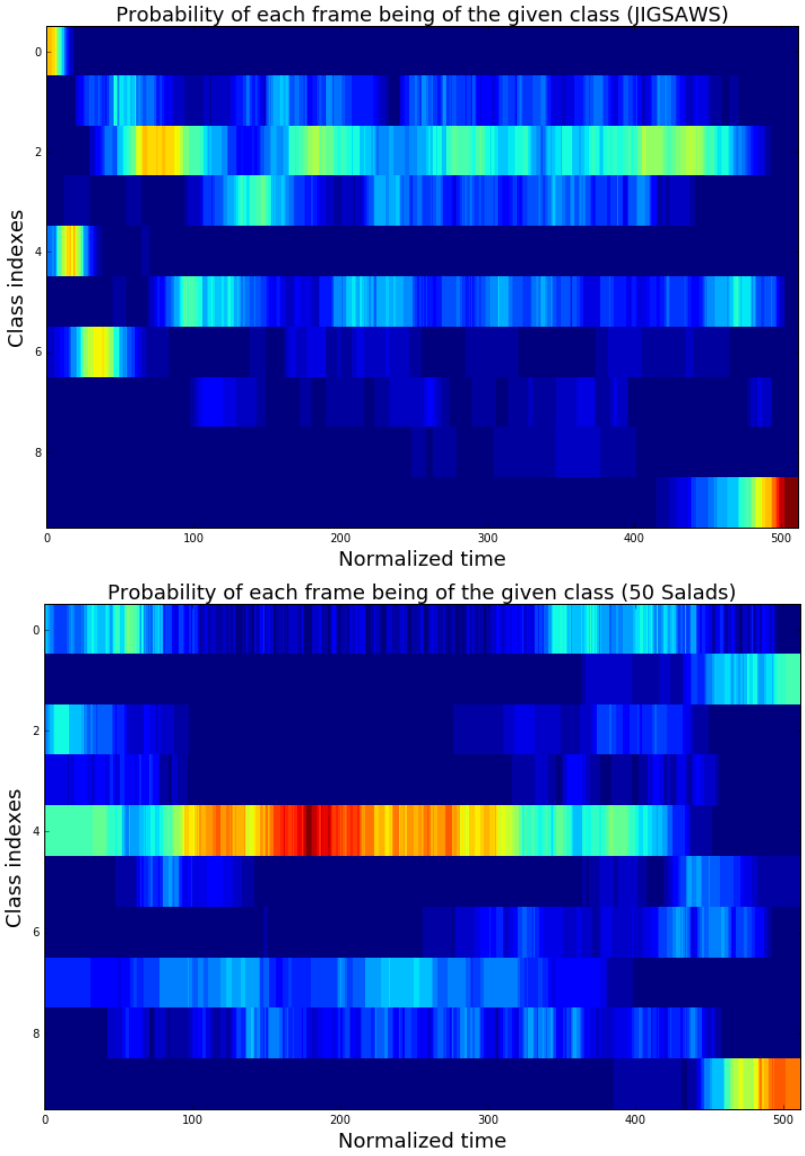


Figure 3.5: The probability of each time step being of each class in the (top) JIGSAWS and (bottom) 50 Salads (higher-level) datasets. Colors go from blue (0%) to green to red (100%).

For every time step, we will add a score corresponding to the likelihood of a given class happening at that time using weight matrix $V \in \mathbf{R}^{C \times T_o}$. The prior score π is given by

$$\pi_s(y, t) = V_{y_t, \tau_t}. \tag{3.5}$$

Note that this prior relies on knowing the total number of time steps in the sequence, and thus is only applicable in offline settings.

Inference

In contrast to n-th order models, exact inference in an SC-CRF is very efficient. Notice that only every d-th time step is connected in our scoring function, which implies we have a set of d independent chains. The best labeling $\hat{y} = \arg\max_y E(X, y)$ can be computed separately per-chain and the resulting predictions can be interlaced. More specifically, if $y_{i:d:T}$ refers to labels at intervals of d (e.g. i, $d + i$, $2d + i$, $\dots$) then $\hat{y}_{i:d:T} = \arg\max_y E(X_{i:d:T}, y_{i:d:T})$.

For each individual skip chain we use the traditional Viterbi algorithm [103]. This is the same algorithm typically used to compute the optimal labeling for HMMs and Linear Chain CRFs. During inference, we compute score $S_{t,y}$ for each class label y at all times t. S is a table of size $T \times C$ where T is the total time and C is the number of classes. This is a dynamic programming problem where, in the forward pass, we

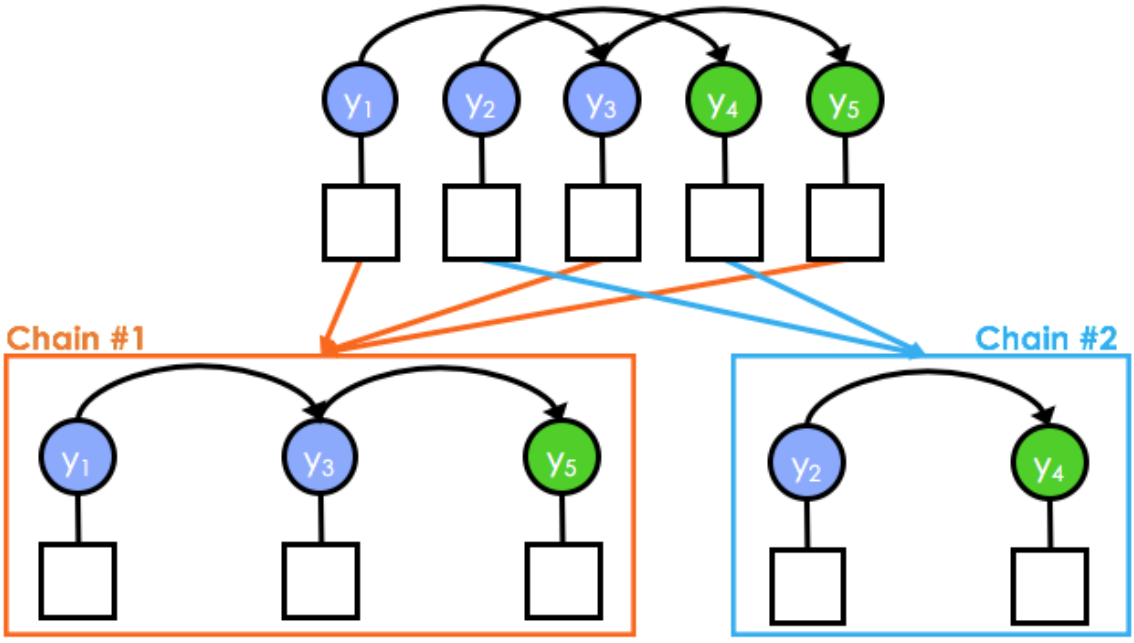


Figure 3.6: Our Skip Chain CRF decomposes time-series data into a set of independent chains. Inference can be performed independently on each chain.

compute

$$S_{t,y} = \begin{cases} \max_{y'} S_{t-d,y'} + W_y X_t + U_{y',y} + V_{y,\tau_t} & \text{if } t > d \\ W_y X_t + V_{y,\tau_t} & \text{otherwise} \end{cases} . \tag{3.6}$$

We output the best label sequence $\hat{y}$ by backtracking through the score table. Pseudo-code for the forward and backward passes is shown in Algorithm 1. The computational complexity is on the order of $O(TC^2)$ operations and $O(TC)$ memory.

While performance using skip chains is superior to linear chains, as we show later, they are prone to over-segmentation. Each pair of sequential time steps is independent thus action predictions may fluctuate spuriously from frame t to $t+1$. We remedy this

Algorithm 1 Viterbi algorithm on skip chains

procedure SkipChainViterbi(X, d, T_o, W, U, V, T, C)

Initialize $S \in \mathbf{R}^{T \times C}$ ▷ Score matrix
Initialize $B \in \{1, \ldots, C\}^{T \times C}$ ▷ Back trace matrix
Initialize $P \in \{1, \ldots, C\}^{T}$ ▷ Output path

Forward Pass: Initialize scores for the first d frames
for $t = 1 : d$ **do**
  for $c = 1 : C$ **do**
    $\tau_t = \text{floor}(\frac{t}{T} T_o)$
    $S_{t,c} = W_c X_t + V_{c,\tau_t}$

Forward Pass: Compute subsequent scores with skip-chain pairwise term
for $t = d + 1 : T$ **do**
  $\tau_t = \text{floor}(\frac{t}{T} T_o)$
  for $c = 1 : C$ **do** ▷ Loop over current class
    $s_{best}, b_{best} = -\infty$, -1 ▷ Init best score and index
    for $c_{prev} = 1 : C$ **do** ▷ Loop over incoming class
      $s_{prev} = S_{t-d,c_{prev}} + W_c X_t + U_{c_{prev},c} + V_{c,\tau_t}$ ▷ Update score
      if $s_{prev} > s_{best}$ **then**
        $s_{best}, b_{best} = s_{prev}, c_{prev}$
    $(S_{t,c}, B_{t,c}) = (s_{best}, b_{best})$ ▷ Update with best score and index

Backward Pass: Retrieve path by back tracing
for $t = T : T - d + 1$ **do**
  $P_t = \arg\max_c S_{t,c}$ ▷ Get the best class for the last d steps
for $t = T - d : 1$ **do**
  $P_t = B[t + d, P_{t+d}]$ ▷ Get the incoming class from back trace.

return P

in two ways. First, we apply a median filter, with a width that is half the length of an action primitive, to the output predictions. Second, in subsection 3.3, we develop a more principled approach by generalizing inference in a Semi-Markov Conditional Random Field to the skip chain model. We find that both solutions have a large decrease in the number of spurious false-positives.

Learning

The parameters of a CRF are often learned using a probabilistic formulation, in which weights are updated to maximize the conditional likelihood of the model [66]. These approaches use gradient descent where each update is based on the marginal distribution of each training sample. In the early 2000s, there were several efforts to learn the parameters of a graphical model using max-margin methods. These solutions typically only required computing the most likely prediction – e.g., using the Viterbi algorithm in a linear chain model – to compute the gradient. In our work we use the Structural Support Vector Machine (SSVM) [104], which has been shown to achieve superior performance compared to probabilistic approaches and to other max-margin approaches [105, 106]. The SSVM is a generalization of the common Support Vector Machine to structured-output problems. For a recent overview on these models and methods see the monograph by Nowozin and Lampert [107].

In order to leverage the SSVM, we must frame the model as a linear function of a set of weights and a set of features. Note that the scoring function we defined before

can already be viewed as a linear function of weights $\theta = vec([W, U, V]) \in \mathbf{R}^{C \cdot F_o}$ and a feature function $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}^{C \cdot F_o}$, where $F_o = F + C + T_o$ is the dimensionality of all concatenated features and C is the number of classes

$$E(X, y) = \theta^\top \Psi(X, y). \tag{3.7}$$

The feature function takes data X and labels y as input and outputs a feature vector which, in our case, is a function of the unweighted and vectorized scoring terms. Note that the output is very sparse; there are only features in the indices corresponding to the label y_t for each time step. We use the notation $\bigoplus_{c=1}^{C} W_c = [W_1^\top, W_2^\top, \ldots, W_c^\top]^\top$ to denote the concatenation of a set of vectors W_c from 1 to C. We use $\mathbf{e}_F[c]$ to denote the vector of length F with all zeros except for a one in the c-th index and $\mathbf{1}[y_t = c]$ denote a scalar which is 1 when y_t is c and 0 otherwise. The function $vec(\cdot)$ vectorizes a matrix. For the SC-CRF, our feature function is

$$\Psi(X, y) = \sum_{t=1}^{T} \begin{bmatrix} \bigoplus_{c=1}^{C} X_t \cdot \mathbf{1}[y_t = c] \\ vec(\mathbf{e}_C[y_{t-d}]\mathbf{e}_C[y_t]^\top \mathbf{1}[t > d]) \\ vec(\mathbf{e}_C[y_t]\mathbf{e}_{T_o}[\tau_t]^\top) \end{bmatrix}. \tag{3.8}$$

To be clear, the first term is of length $F \cdot C$ with zeros everywhere except when y_t is class c. The second (pairwise transition) term is a vectorized version of the $C \times C$ one-hot matrix where index (y_{t-d}, y_t) is one and all other elements are zero. The

temporal prior is a vectorized matrix of size $C \times T_o$ where the index closest in time to (y_t, τ_t) is one and the rest are zeros. By definition, Ψ is a large vector of size $FC + C^2 + CT_o$, however, in practice, only entries indexed by t (e.g., y_t, y_{t-d}, and τ_t) need to be evaluated.

The optimization problem associated with training an SSVM is given by the following objective where $X^{(i)}$ and $y^{(i)}$ are sample sequences from our training set from $i = 1$ to N:

$$\frac{1}{2}\|\theta\|^2 + \frac{C}{N}\sum_{i=1}^{N} \max_{\hat{y}\in\mathcal{Y}} \Delta(\hat{y}, y^{(i)}) + \theta^\top \big(\Psi(X^{(i)}, \hat{y}) - \Psi(X^{(i)}, y^{(i)})\big). \tag{3.9}$$

In this equation, the $\|\theta\|$ term is a regularizer, $\mathcal{Y}$ is the set of all possible labelings for all time-steps and $\Delta(\hat{y}, y^{(i)})$ is a loss function that compares arbitrary labeling $\hat{y}$ with the ground truth labeling $y^{(i)}$ for the ith sequence. We use the Hamming loss

$$\Delta(\hat{y}, y^{(i)}) = \sum_{t=1}^{T} |\hat{y}_t - y_t^{(i)}|, \tag{3.10}$$

which is the sum of the incorrect entries across time steps. The goal is to maximize the margin between the score for the predicted labeling and the ground truth labeling. For certain loss functions, like Hamming, the maximal $\hat{y}$ can be computed efficiently using loss-augmented inference. For more details see [107].

We optimize the parameters of our model using Stochastic Gradient Descent (SGD) where the step size is computed dynamically with Adagrad [108]. Adagrad is

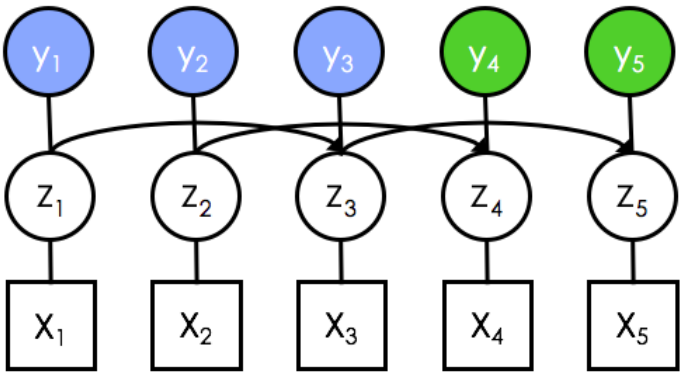


Figure 3.7: Latent Skip Chain Conditional Random Field with a Skip Length parameter $d = 2$.

a recent method, often used in the deep learning literature, which differs from traditional SGD approaches in two ways. First, it defines a separate step size for each parameter in the model as opposed to one parameter per gradient update. Second, it updates each step based on the local history of gradients as opposed to using a constant cooling rate. Notice that, technically speaking, we are computing a subgradient since the objective function is not differentiable, however, it is still typically referred to as SGD.

3.2.2 Latent Skip Chain CRF

In this subsection, we describe a latent variant of the Skip Chain CRF, as depicted in Figure 3.7. Each latent variable can capture stylistic variations on how an action is performed or capture different sub-actions within a larger action. We define these latent states such that each class has its own unique set of latent states.

Let the tuple $z_t = (y_t, h_t)$ be a pairing of a class y_t and discrete latent state

$h_t \in \{1, \ldots, H\}$ for time t and number of latent states per class H. While each class could be associated with a different number of latent states, we assume that the value of H is the same for all classes. Let $h = \{h_t\}_{t=1}^{T}$ denote the sequence of all latent states irrespective of the class labels. Our new scoring function is

$$E(X, y) = \max_{h} \sum_{t=1}^{T} \phi(X, (y, h), t) + \psi((y, h), t) + \pi((y, h), t). \tag{3.11}$$

Our new terms are a function of both class y_t and that class's hidden state h_t with parameters $W \in \mathbf{R}^{(C \cdot H) \times F}$, $U \in \mathbf{R}^{(C \cdot H) \times (C \cdot H)}$, and $V \in \mathbf{R}^{(C \cdot H) \times T_o}$:

$$\textbf{Frame-wise Unary: } \phi(X, z, t) = W_{z_t} X_t \tag{3.12}$$

$$\textbf{Skip Chain Pairwise: } \psi(z, t) = U_{z_{t-d}, z_t} \mathbf{1}[t > d] \tag{3.13}$$

$$\textbf{Temporal Prior: } \pi(z, \tau_t) = V_{z_t, \tau_t}. \tag{3.14}$$

Note that W_{z_t} refers to $W_{y_t \cdot H + h_t}$; there is a block of H consecutive indices corresponding to each class. This holds true for W, U, and V.

Latent states are defined per-class, so inference is almost identical to that of the SC-CRF. The sequence z is decoded using the Viterbi algorithm from Algorithm 1 and then the sequence of labels y is obtained directly from z. Note that while this does increase the number of parameters by a factor of H, the values of C and H are low enough (e.g. $C = 10$, $H = 3$) that the computational complexity is still manageable on any modern computer. The product of all classes and latent states is

$Z = C \cdot H$.

We use the Latent Structural Support Vector Machine with the Concave Convex Procedure (CCCP) [96], using the approach described in Nowozin and Lampert [107] (Algorithm 16), to learn the parameters of the latent skip chain model. CCCP alternates between updating the hidden states h_t at each time step and updating the weights using gradient descent. Again, we use Stochastic Gradient Descent with Adagrad. Our new feature function is defined as

$$\Psi(X, z) = \sum_{t=1}^{T} \begin{bmatrix} \bigoplus_{c=1}^{C} \Big(\bigoplus_{h=1}^{H} X_t \cdot \mathbf{1}[z_t = (c, h)] \Big) \\ vec(\mathbf{e}_Z[z_{t-d}]\mathbf{e}_Z[z_t]^\top)\mathbf{1}[t > d] \\ vec(\mathbf{e}_Z[z_t]\mathbf{e}_{T_o}[\tau_t]^\top) \end{bmatrix}, \tag{3.15}$$

where all of the terms are now operating over z rather than y. The length of Ψ, and correspondingly the number of parameters in θ, is $Z \cdot (F + Z + T_o)$.

It is important to initialize our weights before solving the LSSVM, as highlighted by Pirsiavash and Ramanan [57] and Tang *et al.* [69]. The learning process only updates hidden states that are in use, so if we do not initialize properly then we may learn "dead" states which are never activated.[1] We initialize the latent unary terms in our model using a KMeans-based procedure similar to Tang *et al.*for their discriminative hidden semi-Markov model. Procedurally, we first divide all action

[1]For reference see lines 11 and 13 in the CCCP routine in Nowozin and Lampert [107] (Algorithm 16). If a given latent state ($\hat{z}^n$ in their notation) is never selected in the max operator then its weights will never be updated.

instances of a given class into H subsegments corresponding to the H latent variables per class. Each subsegment used for KMeans must be the same duration, so we resample each segment to have the same duration T_k. These subsegments, now of size $F \times T_k$, are flattened into vectors of length $F \cdot T_k$, and input into KMeans. We apply KMeans and set the weights for each latent state of the given class, W_z, to be one of the means.[2] This procedure is applied independently for each class. Qualitatively, we find that this results in fewer "dead" states which are never activated. Note that a potentially better alternative would be to initialize using the approach of Lobel *et al.* [109] which computes clusters using KMeans and then trains a multi-class SVM for each cluster. For all other scoring terms we initialize parameters using random orthonormal matrices, for example as described in [110].

3.2.3 Latent Convolutional CRF

In the previous section we assumed that each unary term was simply a function of the data at each individual frame. However, we know that the values of the input features (e.g., accelerometers) vary substantially within each action instance. For example, as a user chops a knife the corresponding accelerometer will oscillate as the use moves their hand up and down. In this section we describe a unary term that captures how sensor values change over the course of a long temporal window. Our

[2]Note that these means are not intended to correspond to the exact coefficients of the classifiers. The goal is to initialize weights such that after learning two states do not have weights that are effectively the same.

approach contrasts with hand-crafted approaches (e.g. [4, 22, 25, 5, 12]) which tend to employ application-specific features, like object-specific region detectors from RGBD video [5], and often do not generalize across domains or even environmental setups. We introduce a type of action primitive that uses convolutional filters to model how the input sensor signals change over the course of each action. We show experiments using one filter per class and a variant using latent action primitives. Figure 3.8 highlights the difference between frame-wise unary weights, our action primitives, and our latent action primitives.

Convolutional Action Primitives

In the first approach, each action class $y \in \mathcal{Y}$ is represented by a single convolutional filter where the row in each filter corresponds to the features at each time step within an action. Ideally, the length of each filter should match the length of a predicted segment, however, this is problematic because the duration of each action varies substantially.[3]

We denote the collection of filters $W = \{W^c\}_{c=1}^{C} \in \mathbf{R}^{C \times d \times F}$, where the filter for the c-th class is $W^c \in \mathbf{R}^{d \times F}$. Each filter has a corresponding bias such that $b = \{b_c\}_{c=1}^{C}$ with $b_c \in \mathbf{R}$. First, let us define the convolution operator $W * X$ which

[3]We could choose to have filters of many different lengths. However, we define all filters to be of the same length in efforts to reduce the number of hyper-parameters.

has entries

$$(W * X)_{c,t} = \sum_{j=1}^{F} \sum_{t=1}^{d} W_{t,j}^{c} X_{j,t-t+1}. \tag{3.16}$$

We want both the input and output lengths be of size T so we zero pad X such that $X_{j,t} = 0 \quad \forall t \leq 0, 1 \leq j \leq F$. As a result the output of $W * X$ is a matrix of size $T \times C$.

The score for this component of our model is given using the convolution operator is

$$\phi(X, y, t) = (W * X)_{y_t,t} + b_{y_t}. \tag{3.17}$$

We jointly learn these filters with the Skip Chain CRF and name this model the Convolutional Skip Chain CRF (CSC-CRF). Note that the features are now a function of time steps $t - d$ to t, so we use the notation $X_{t-d:t}$ to indicate the data over the last d time steps. The feature function Ψ is

$$\Psi(X, y, t) = \sum_{t=1}^{T} \begin{bmatrix} \bigoplus_{c=1}^{C} vec(X_{t-d+1:t}) \cdot \mathbf{1}[y_t = c] \\ vec(\mathbf{e}_C[y_{t-d}]\mathbf{e}_C[y_t]^\top)\mathbf{1}[t > d] \\ vec(\mathbf{e}_C[y_t]\mathbf{e}_{T_o}[\tau_t]^\top) \end{bmatrix}. \tag{3.18}$$

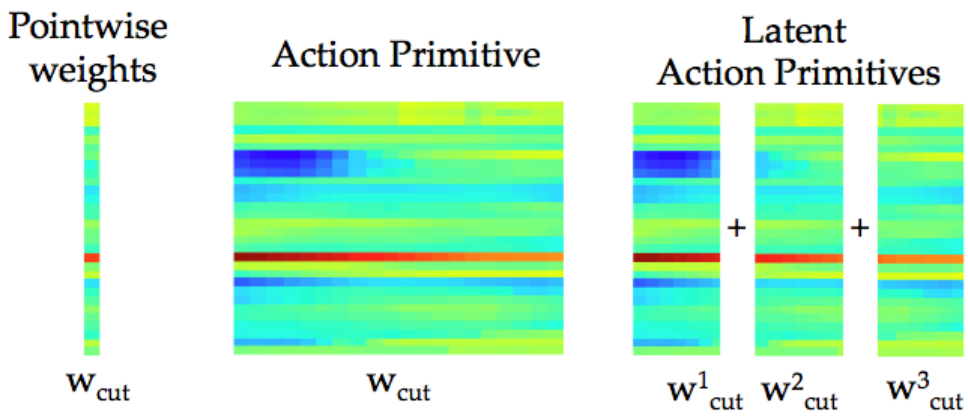


Figure 3.8: Action primitives for the class *cutting* in the 50 Salads dataset. Each row corresponds to weights for the X, Y, or Z axis of an accelerometer over time. Red is high, green is neutral, and blue is low. (left) traditional weight vector applied to a single frame (middle) our convolutional action primitives (right) and our latent action primitives.

Latent Convolutional Action Primitives

In practice, the duration of each action instance may differ substantially between users. For example, in a *cutting* action, one person may pause between picking up a knife and cutting a vegetable. In addition, users may perform actions in different styles or orderings. Thus, it may be advantageous to learn a separate filter for each part of an action, such as the start, middle, and end. We use latent variables to learn a set of subactions for each action class. Recall that these subactions are learned in an unsupervised manner based on the higher-level action labels. They are initialized by splitting actions into different partitions but may take on other latent meanings.

Recall, $z_t = (y_t, h_t)$ is the latent state and corresponding class at time t. We define a new set of filters $W = \{W^{(c,h)}\}_{c=1,h=1}^{C,H}$ corresponding to all latent states h and classes c, which results in $Z = C \cdot H$ filters. Again, we assume that each action

has the same number of subactions H. The score for any hidden state is:

$$\phi(X, z, t) = (W * X)_{z_t,t} + b_{z_t}. \tag{3.19}$$

We learn the action primitives jointly with the Latent SC-CRF to form the Latent Convolutional Skip Chain CRF (LC-SC-CRF). Using the same notation as before, the feature function Ψ is

$$\Psi(X, z, t) = \sum_{t=1}^{T} \begin{bmatrix} \bigoplus_{h=1}^{H} \left(\bigoplus_{c=1}^{C} vec(X_{t-d+1:t}) \cdot \mathbf{1}[z_t = (c, h)] \right) \\ vec(\mathbf{e}_Z[z_{t-d}]\mathbf{e}_Z[z_t]^\top)\mathbf{1}[t > d] \\ vec(\mathbf{e}_Z[z_t]\mathbf{e}_{T_o}[\tau_t]^\top) \end{bmatrix} \tag{3.20}$$

. This model is learned in the same way as the Latent CRF previously described using the following scoring function, which maximizes over the best scoring filters h_t

$$E(X, y) = \max_h \sum_{t=1}^{T} \theta^\top \Psi(X, (y, h), t), \tag{3.21}$$

In Section 5.4 we compare the frame-wise, action primitive, and latent action primitive models. These are referred to as SC-CRF, LC-SC-CRF ($H = 1$), and LC-SC-CRF ($H > 1$) respectively. See Figure 3.9 for the final depictions of each convolutional model.

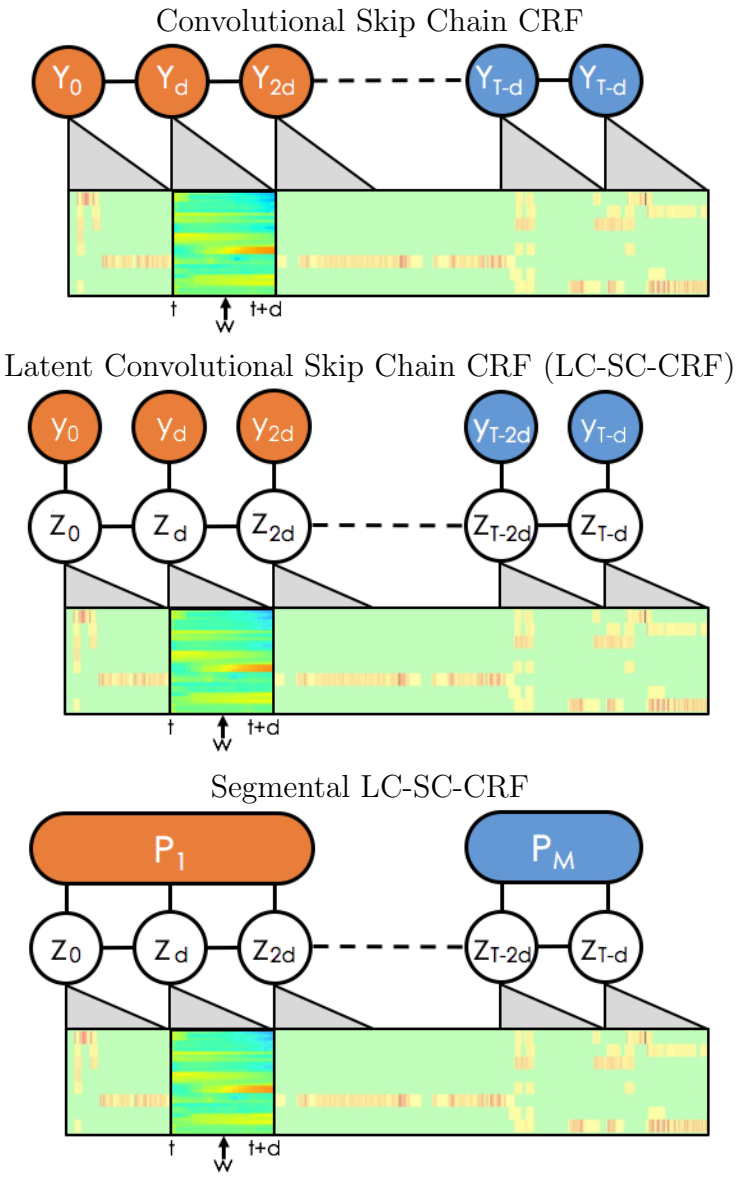


Figure 3.9: Convolutional variants of each model.

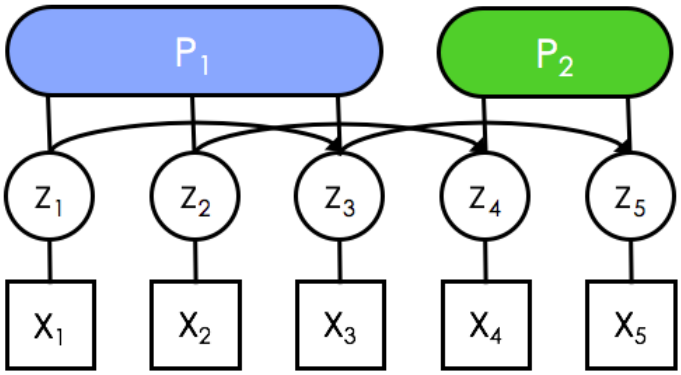


Figure 3.10: Segmental Latent Skip Chain Conditional Random Field with a Skip Length parameter $d = 2$.

3.3 Segmental Inference

As previously mentioned, each chain in our model is inferred independently which can result in spurious fluctuations between action labels across a short sequence of time steps. This will become apparent when we visualize our results later. In this section, we describe a segmental formulation of our model, as depicted in Figure 3.10, and a corresponding inference algorithm, which reduces these oversegmentation issues. For the sake of simplicity, we describe the segmental inference procedure for the model without latent variables (Equation 3.8) and then mention how to generalize it to the latent case (Equations 3.15 and 3.20).

There exist many segmental models in the literature, such as those proposed by Sarawagi and Cohen [71] for Semi-Markov CRFs, by Shi *et al.* [75] for discriminative semi-Markov models and by Pirsiavash and Ramanan [57] for Segmental Regular Grammars. One limitation of many of these models is that their computational com-

plexity is much higher than their frame-wise alternatives. In this section we propose an inference algorithm that is one to three orders of magnitude faster than the current inference technique using the datasets we evaluate on. Note that Titsias *et al.* [111] introduced the K-Segment HMM around the same time as we published [112]. Their model uses a similar idea as ours and thus has the same computational complexity. One difference is that they assume there are exactly K segments, whereas we allow for a variable number segments up to an upper bound.

We start with notation equivalent to Sarawagi and Cohen [71]. Let tuple $P_j = (\mathbf{y}_j, t_j, l_j)$ be the jth action segment, where $\mathbf{y}_j$ is the action label, t_j is the start time, and l_j is the segment duration. There is a variable number of segments, M, so that the sequence of action segments is given by $P = \{P_1, \ldots, P_M\}$ for $0 < M \leq T$. In the same way that sequence $y = \{y_t\}_{t=1}^{T}$ denotes the per-frame labels, $\mathbf{y} = \{\mathbf{y}_j\}_{j=1}^{M}$ notes the per-segment labels. Given segment labels, start times and segment durations, one can compute frame labels as

$$y_t = \mathbf{y}_j \text{ for } t_j \leq t < t_j + l_j. \tag{3.22}$$

Note that the start of segment j coincides with the end of the previous segment, i.e. $t_j = t_{j-1} + l_{j-1}$, and the durations sum to the total time $\sum_{i=1}^{M} l_i = T$. We also assume that the duration is bounded by a minimum segment length l_{min} and a maximum length l_{max}. We infer segments P that maximize the total score $E(X, P)$ for the data

using a segment function $f(\cdot)$ where

$$E(X, P) = \sum_{j=1}^{M} f(X, \mathbf{y}_{j-1}, \mathbf{y}_j, t_j, l_j). \tag{3.23}$$

The segment function will be slightly different depending on which of our previously described scoring functions is used. Here we will describe the linear chain case with a temporal prior and without latent states. The segment function is the sum of the per-frame unary, pairwise, and temporal prior scores

$$f(X, \mathbf{y}_{j-1}, \mathbf{y}_j, t_j, l_j) = \sum_{t=t_j}^{t_j+l_j-1} \begin{cases} W_{\mathbf{y}_t} X_t + U_{y_{t-1}, y_t} + V_{\mathbf{y}_t, \tau_t} & \text{if } t-1 > 0 \\ W_{\mathbf{y}_t} X_t + V_{\mathbf{y}_t, \tau_t} & \text{otherwise.} \end{cases} \tag{3.24}$$

Recall that y_t was defined in Equation 3.22. Note in the original Semi-Markov CRF formulation [71] there was an additional segmental pairwise term. In our model, this is modeled using the sum of the pairwise terms within a segment. As such, our energy is the same as in Equation 3.1 (with $d = 1$). The difference comes from the constraints that we add during inference.

Segmental Inference

The traditional semi-Markov inference method [71] solves the following discrete optimization problem

$$\max_{P \in \mathcal{P}} \; E(X, P) \;\; s.t. \quad t_j = t_{j-1} + l_{j-1} \; \forall j \;\; \text{and} \;\; l_{min} \leq l_j \leq l_{max} \; \forall j \;\; \text{and} \;\; \sum_{i=j}^{M} l_j = T, \tag{3.25}$$

where $\mathcal{P}$ is the set of all segmentations and l indicates the duration of a segment. The constraints in this problem prevent usage of the traditional Viterbi algorithm. The optimal labeling is typically found using an extension of the Viterbi algorithm to the semi-Markov case, which we refer to as Segmental Viterbi [71, 75, 57]. The algorithm recursively computes the score $S_{t,c}$ for the best labeling whose last segment ends at time t and is assigned class c:

$$S_{t,c} = \max_{\substack{l \in \{1 \ldots L\} \\ c' \in \mathcal{Y} \setminus c}} \begin{cases} S_{t-l,c'} + f(X, c', c, t-l, l) & \text{if } t - l > 0 \\ f(X, c', c, 1, t) & \text{otherwise.} \end{cases} \tag{3.26}$$

As with the Viterbi algorithm, the optimal labels are recovered by backtracking through the matrix S using the predicted segment durations.

This approach is inherently frame-wise: for each frame, compute scores for all possible segment durations, current labels, and previous labels up to frame t. This results in an algorithm of complexity $O(T^2C^2)$, in the naive case, because the dura-

tion of each segment ranges from 1 to T. If the segment duration is bounded then complexity is reduced to $O(TC^2(l_{max} - l_{min}))$ [71, 75, 57]. The Segmental Viterbi algorithm with duration constraints l_{min} and l_{max} is shown in Algorithm 2.

To further accelerate the computation of the optimal labels, we introduce an alternative approach in which we constrain the number of segments, M, by an upper bound, K, such that $0 < M \leq K$. If $K = T$, this is of the same complexity as Segmental Viterbi. We will no longer need duration variables l_j, which are redundant given all times t_j, so we simplify the segment notation to be $\hat{P}_j = (\mathbf{y}_j, t_j)$. Now, instead of adding constraints on the durations of each segment, we only require that the start of the jth segment comes after segment $j - 1$. We solve the problem

$$\max_{\substack{M \in \{1,\ldots,K\} \\ \hat{P} \in \hat{\mathcal{P}}_M}} E(S, \hat{P}) \quad s.t. \quad t_{j-1} < t_j \ \forall j \in \{1, \ldots, M\}. \tag{3.27}$$

where $\hat{\mathcal{P}}_M$ is the set of all segmentations with M segments. Our approach has two advantages: it substantially decreases computation time and it prevents gross over-segmentation because there can never be more than K segments. While this does introduce a new parameter which may need to be tuned, in practice we set K to be the maximum number of segments in the training set.

In our model the score for each segment is the sum of scores over each frame, so the total score for the segmentation containing k segments can be recursively computed using the scores for a segmentation containing $(k - 1)$ segments. Specifically, we first

Algorithm 2 Segmental Viterbi [71] (with duration constraints)

procedure SEGMENTALVITERBI(X, l_{min}, l_{max}, T_o, W, U, T, C)

Initialize $S \in \mathbf{R}^{T \times C}$ ▷ Score matrix
Initialize $I \in \mathbf{R}^{T \times C}$ ▷ Integral score matrix
Initialize $B \in \{1, \ldots, C\}^{T \times C}$ ▷ Back trace matrix
Initialize $D \in \mathbf{Z}^{T \times C}$ ▷ Duration matrix
Initialize $P \in \{1, \ldots, C\}^{T}$ ▷ Output path

Forward Pass: Compute integral image for efficient score updates
for $c = 1 : C$ **do**
  $I_{t,c} = (W_c X_t + V_{c,0})^\top$
for $t = 1 : l_{max}$ **do**
  $\tau_t = \text{floor}(t \frac{T_o}{T})$
  for $c = 1 : C$ **do**
    $I_{t,c} = I_{t-1,c} + (W_c X_t + V_{c,\tau_t})^\top$
▷ For efficient segment scores

Forward Pass: Compute scores for the first segment
for $t = 1 : l_{max}$ **do**
  $(S_{t,\cdot}, D_{t,\cdot}) = (I_{t,\cdot}, t)$

Forward Pass: Compute scores for the subsequent segments
for $t = l_{min} + 1 : T$ **do** ▷ Compute score for segment ending at every time
  $\tau_t = \text{floor}(t \frac{T_o}{T})$
  for $c = 1 : C$ **do** ▷ Current class
    $(s_{best}, b_{best}, d_{best}) = (-\infty, -1, -1)$ ▷ Init best score, class index, & duration
    for $c_{prev} \in \mathcal{Y} \backslash c$ **do** ▷ Class of previous segment
      for $d = l_{min} : l_{max}$ **do** ▷ Maximize over segment durations
        $s_{prev} = S_{t-d,c_{prev}} + U_{c_{prev},c} + I_{t,c} - I_{t-d,c}$ ▷ New segment score
        if $s_{prev} > s_{best}$ **then**
          $s_{best}, b_{best}, d_{best} = s_{prev}, c_{prev}, d$
    $S_{t,c}, B_{t,c}, D_{t,c} = s_{best}, b_{best}, d_{best}$ ▷ Keep best score, index, duration

Backward Pass: Retrieve the best segments by back tracing
$c = \arg\max_c S_{t,c}$ ▷ Get the best class for the last segment
$d = D_{t,c}$ ▷ Best segment duration
$P_{T-d+1:T} = c$ ▷ Set all steps within a segment to class c
$t = T - d$
while $t > 1$ **do**
  $c = B_{t+d,c}$ ▷ Get the incoming class
  $d = D_{t,c}$
  $P_{t-d+1:t} = c$
  $t = t - d$

return P

compute the best segmentation assuming $M = 1$ segments, then compute the best segmentation for $M = 2$ segments, up to $M = K$ segments. Let $\bar{S}^k_{t,c}$ be the score for the best labeling with k segments ending in class c at time t. The scores for the first segment are given by

$$\bar{S}^1_{t,c} = \begin{cases} \bar{S}^k_{t-1,c} + U_{c,c} + W_c X_t + V_{c,\tau_t} & \text{if } t > 1 \\ W_c X_t + V_{c,\tau_t} & \text{otherwise} \end{cases} \tag{3.28}$$

and the score for each subsequent sequence is

$$\bar{S}^k_{t,c} = \max \left\{ \max_{c' \in \mathcal{Y} \setminus c} (\bar{S}^{k-1}_{t-1,c'} + U_{c',c}), \quad \bar{S}^k_{t-1,c} + U_{c,c} \right\} + W_c X_t + V_{c,\tau_t}. \tag{3.29}$$

Note that the second segment must come after the first segment, which implies that you cannot be in the second segment in the first frame. Thus, $\bar{S}^k_{t,c} = -\infty \quad \forall t \in \{1, \ldots, k-1\}$. The recursion in the latter equation contains two cases: (1) if transitioning into a new segment ($c' \neq c$), use the best incoming score from the previous segment $k-1$ at $t-1$ and (2) if staying in the same segment ($c' = c$), use the score from the current segment at $t-1$. Our forward pass, in which we compute each score $\bar{S}^k_{t,c}$, is shown in Algorithm 3. The optimal labeling is found by backtracking through $\bar{S}$.

If $L = l_{max} - l_{min}$, the complexity of our algorithm, $O(KTC^2)$, is $\frac{L}{K}$ times more efficient than Segmental Viterbi assuming $K < L$. In most practical applications, K

Algorithm 3 K-Segment Viterbi (ours)

procedure KSEGMENTVITERBI(X, K_{max}, W)

Initialize $S \in \mathbf{R}^{K_{max} \times T \times C}$ ▷ Score tensor

Initialize $B \in \{1, \ldots, C\}^{K_{max} \times T \times C}$ ▷ Back trace tensor

Initialize $D \in \mathbf{Z}^{K_{max} \times T \times C}$ ▷ Duration tensor

Initialize $P \in \{1, \ldots, C\}^{T}$ ▷ Output path

Forward Pass: Compute scores for the first segment

for $t = 1 : T$ **do**

$\tau_t = \text{floor}(t \frac{T_o}{T})$

$S^1_{t,\cdot} = S^1_{t-1,\cdot} + (WX_t + V_{\cdot,\tau_t})^\top$

$D_{1,t,\cdot} = t$

Forward Pass: Compute scores for the $(m-1)$ subsequent segments

for $m = 2 : K_{max}$ **do**

for $t = m : T$ **do** ▷ Transition to new segment at t

$\tau_t = \text{floor}(t \frac{T_o}{T})$

for $c = 1 : C$ **do** ▷ Current class

$(s_{best}, b_{best}, d_{best}) = (S_{m,t-1,c}, B_{m,t-1,c}, D_{m,t-1,c} + 1\ )$ ▷ Same segment

for $c_{prev} \in \mathcal{Y} \backslash c$ **do** ▷ Previous class

if $c_{prev} \neq c$ **then**

$s_{prev} = S^{m-1}_{t-1,c_{prev}} + U_{c_{prev},c}$ ▷ Transition to new segment

else

$s_{prev} = S^{m}_{t-1,c_{prev}} + U_{c_{prev},c}$ ▷ Stay in the same segment

if $s_{prev} > s_{best}$ **then**

$(s_{best}, b_{best}, d_{best}) = (s_{prev}, c_{prev}, 1)$

$S^m_{t,c} = s_{best} + (W_c X_t + V_{c,\tau_t})^\top$ ▷ Keep best score, update with new time step

$(B_{t,c}, D_{t,c}) = (b_{best}, d_{best})$ ▷ Keep best index and duration

Backward Pass: Retrieve the best segments by back tracing

$m_{segs} = \arg\max_m \max_c S_{m,T,c}$ ▷ Find optimal segment count

$c = \arg\max_c S_{m_{segs},T,c}$ ▷ Get the best class for the last segment

$d = D_{m_{segs},T,c}$ ▷ Best segment duration

$P_{T-d+1:T} = c$ ▷ Set all steps within a segment to class c

$t = T - d$

$m = m_{segs}$

while $m > 1$ **do**

$c = B_{t+d,c}$ ▷ Get incoming class

$m = m - 1$

$d = D_{m,t,c}$

$P_{t-d+1:t} = c$

$t = t - d$

return P

is much smaller than L. In the evaluated datasets there is a speedup of one to three orders of magnitude. Note, however, our method requires K times more memory than Segmental Viterbi. Ours has space complexity $O(KTC)$, whereas Segmental Viterbi has complexity of $O(TC)$. Typically $K \ll T$ so the increase in memory is easily manageable on any modern computer. In all cases, we set K based on the maximum number of segments in the training split.

The described algorithm was defined for the non-latent version of our model. There are multiple ways of incorporating latent variables into our algorithm. One way is to follow the approach of Andrew [113] for Markov Semi-Markov Model CRFs, which has intra-segment and inter-segment pairwise terms. The parameters of their intra-segment term would correspond to the parameters of our latent pairwise skip-chain term.

3.4 Baselines

In the results section of this chapter we will compare against prior work on JIGSAWS and 50 Salads, and we will add two additional baselines for each dataset. The first is a standard Support Vector Machine (SVM) and the second is a Dynamic Time Warping approach.

The SVM is like our SC-CRF but only consists of a unary term, such that the

feature function is

$$\Psi(X, y, t) = \sum_{t=1}^{T} \Big(\bigoplus_{c=1}^{C} X_t \cdot \mathbf{1}[y_t = c] \Big). \tag{3.30}$$

The class label for each frame is predicted independently. For these results, we use the standard SVM implementation in Scikit-Learn [114].

Dynamic Time Warping and its variants are typically used to align two time series sequences but have also been used to successfully perform action segmentation (e.g., [115, 116, 117, 118, 119]). A common formulation is as follows. For any two sequences indexed by i and j and with sequence lengths $T^{(i)}$ and $T^{(j)}$, we compute an optimal set of correspondences, $r = \{r_t\}_{t=1}^{T^{(i)}}$, such that the data $X_t^{(i)}$ corresponds to the data $X_{r_t}^{(j)}$. Figure 3.11 depicts two 1D signals and their correspondences. In order to compute r, we minimize the error between each time step in $X^{(i)}$ and the corresponding time step in $X^{(j)}$ given a set of constraints

$$\sum_{t=1}^{T^{(i)}} \left\| X_t^{(i)} - X_{r_t}^{(j)} \right\| \quad \text{s.t. } r_t \leq r_{t+1} \quad \& \quad r_1 = 1, r_{T^{(i)}} = T^{(j)}. \tag{3.31}$$

The first constraint ensures monotonicity ($r_t \leq r_{t+1}$), meaning one cannot go backwards in time, and the second constraint ensures that the start and finish time of the

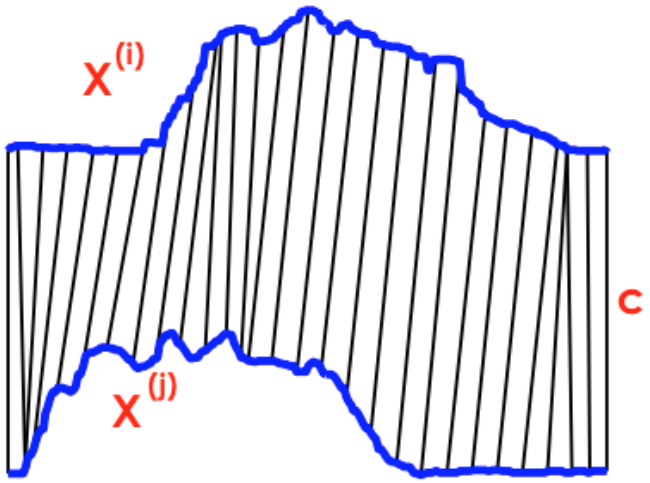


Figure 3.11: Depiction of sample sequences $X^{(i)}$ and $X^{(j)}$ with correspondences c. Modified from [10].

sequences are the same. The scalar DTW distance is given by

$$DTW(X^{(i)}, X^{(j)}, r) = \frac{1}{T^{(i)}} \sum_{t=1}^{T^{(i)}} \left\| X_t^{(i)} - X_{r_t}^{(j)} \right\|. \tag{3.32}$$

In practice, one can compute the DTW distance using dynamic programming [120].

We use the nearest-neighbors algorithm with the DTW distance to find the best matching sequence from our training set.[4] We then propagate the labels from the best sequence to the new test sequences such that $Y_t^{(i)} = Y_{r_t}^{(j)}$ for all t.

DTW is a very effective model on some datasets. In particular, if the order of actions is the same between different sequences, then it may accurately capture how the input sensor signals evolve within and between actions. If the order of actions

[4]In a slight abuse of terminology, we refer to both the metric and the nearest neighbors segmentation model as DTW.

varies substantially between sequences, or if the order in a test sequence has never been seen before, then DTW will likely not perform well.

Depending on the training set size, this method may be slower than our proposed solutions. In the naive case, finding the correspondences has computational complexity $O(T^2)$ for each training sequence. Thus, if there are N training examples, the complexity with nearest neighbors is $O(NT^2)$. Using the efficient lower bound by Keogh and Ratanamahatana [121], it is possible to compute the DTW score without the correspondences in $O(T)$. Thus the DTW-NN algorithm is of complexity $O(NT + T^2)$. It may be possible to compute the correspondences more efficiently using pruning [122]. For comparison, our linear chain CRFs are on the order of $O(TC^2)$, where typically C^2 is much less than T and is independent of the number of training examples.

3.5 Evaluation

In this section we describe our evaluation metrics, datasets, and results.

3.5.1 Metrics

We assess our models using three primary metrics and also compare against prior work using frame-wise precision and recall. In Chapter 5 we will discuss metrics in more depth when we compare action segmentation and action detection.

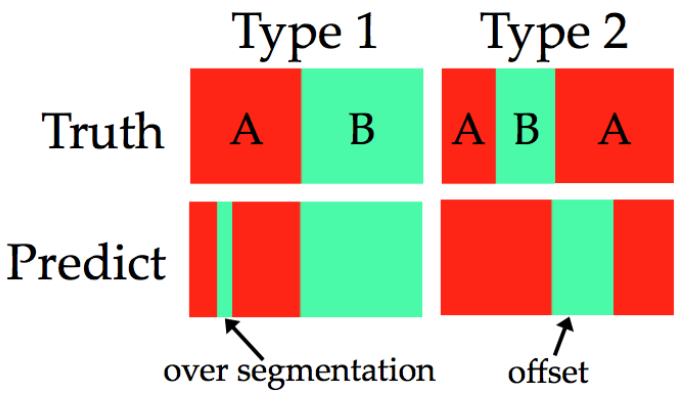


Figure 3.12: Our metrics measure two types of errors. First is over-segmentation, which is when there are multiple predicted action segments contained within one true segment. The second evaluates the sequential ordering of actions and allows for small temporal offsets. The offsets are sometimes caused by inter-reviewer variability and should not negatively impact performance.

The first metric, frame-wise accuracy, is commonly used in robotics and computer vision (e.g., [3, 50, 92]), and gives a sense of the overall correctness of our predictions. The score is $M_{acc}(y, \hat{y}) = \sum_{t=1}^{T} \mathbf{1}[y_t = \hat{y}_t]$ where $y = \{y_t^*\}_{t=1}^{T}$ is the sequence of ground truth labels and $\hat{y} = \{\hat{y}_t\}_{t=1}^{T}$ is the sequence of predictions.

In many applications, like surgical workflow, there is large uncertainty as to when one action stops and another starts due to inter-annotator variability. Similarly in surgical skill evaluation the precise temporal segmentation may not be as important as computing the correct ordering in which actions take place. As such, the second metric measures how well the model predicts the sequential order of action segments, independent of temporal shifts, which we evaluate using a segmental edit score. An example temporal shift is depicted in Figure 3.12 (right). This encodes the order of the actions but does not include the start or stop time of any segment. To compute this score we use notation for segmental inference. Let $P = \{y_j\}_{j=1}^{M}$ be the set of

labels for each of the M segments in a ground truth sequence and $\hat{P} = \{\hat{y}_j\}_{j=1}^{\hat{M}}$ be the set of labels in the predicted sequence. For example, if $y = \{aabbbccccc\}$ then $P = \{abc\}$. Our segmental edit score is defined using a normalized edit distance, $M_{edit}(P, \hat{P})$, with insertions, deletions, and replacements. We compute this using the Levenshtein distance, $L(\cdot)$, as described by Navarro [123], where i and j are indexes within the given true and predicted sequences.

$$\mathrm{L}(P, \hat{P}, i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \begin{cases} \mathrm{L}(P, \hat{P}, i-1, j) + 1 \\ \mathrm{L}(P, \hat{P}, i, j-1) + 1 \\ \mathrm{L}(P', \hat{P}, i-1, j-1) + 1[P_i \neq \hat{P}_j] \end{cases} & \text{otherwise.} \end{cases} \tag{3.33}$$

The score is normalized by taking the maximum length of each sequence such that $M_{edit}(P, \hat{P}) = (1 - \frac{L(P,\hat{P},M,\hat{M})}{\max(M,\hat{M})}) \cdot 100$ where $|\cdot|$ denotes the length of a sequence. A score of 100 is best and 0 is worst.

Our third metric measures how much overlap there is between ground truth and predicted segments. This penalizes oversegmentation errors as depicted in Figure 3.12 (left). This metric computes the intersection over union of the longest contiguous predicted segment for a given ground truth segment. Our score is similar to the

Jaccard Index [124] except it penalizes spurious segments. It is given by

$$M_{overlap}(P, \hat{P}) = \frac{100}{M} \sum_{i=1}^{M} \max_{j} \frac{|P_i \cap \hat{P}_j|}{|P_i \cup \hat{P}_j|}, \tag{3.34}$$

where $|P_i \cap \hat{P}'_j|$ denotes the number of frames in segments P_i and $\hat{P}_i$ that intersect and where $|P_i \cup \hat{P}'_j|$ denotes the number of frames in these segments that overlap. This score lies in $[0, 100]$ where a higher value is better.

3.5.2 Datasets

In this chapter, we evaluate on 50 Salads and JIGSAWS. For 50 Salads we only use accelerometer data. Due to symmetries in the tools we use the unsigned acceleration for each axis (XYZ) as opposed to the signed values. This results in signals with non-negative values where 0 means an object is not accelerating and everything else means that it is accelerating. We evaluate using 5-fold cross validation where we train on 20 users (40 videos) and test on 5 users (10 videos). We set the skip length and filter length to $d = 200$ frames which was determined via cross validation on one of the splits. Recall that 50 Salads has multiple granularities of action labels. Unless otherwise stated, results are on the "eval" graularity, but additional granularities are included in Table 3.7.

On JIGSAWS, we evaluate on the suturing task using Leave One User Out as described in [91]. In each split we train on seven users and test on the left out

Model Name	Abbreviation	Relevant Equation
Skip Chain CRF	SC-CRF	3.8
Latent Skip Chain CRF	LSC-CRF	3.15
Convolutional Skip Chain CRF	CSC-CRF	3.18
Latent Convolutional Skip Chain CRF	LC-SC-CRF	3.20

Table 3.1: We assess performance using four variations on our model.

user. We use a skip length and action primitive duration of $d = 100$ frames unless otherwise stated. We use a subset of the robot kinematics for the left and right (slave-side) end effector from a da Vinci surgical robot. Specifically, the inputs are tool positions, velocities, and normalized gripper angles. We lightly preprocess the data by subtracting the mean (per sequence) and dividing by the per-feature standard deviation.

3.5.3 Experiments

We performed experiments to asses the impact of skip chains, segmental inference, and convolutional filters using four variations on our model, as denoted in Table 3.1. Unless otherwise mentioned, inference is performed using Viterbi (with Skips) and is post-processed using a median filter with width equal to half of the convolutional filter length (or skip length if no filters are used).

In all experiments except the skip length analysis (shown in Figure 3.13) we subsampled the data temporally by 10x for 50 Salads and 5x for JIGSAWS. This level of subsampling had a minimal effect ($\approx 0.1\%$) on accuracy. All parameters indicated are with respect to the original duration of each sequence.

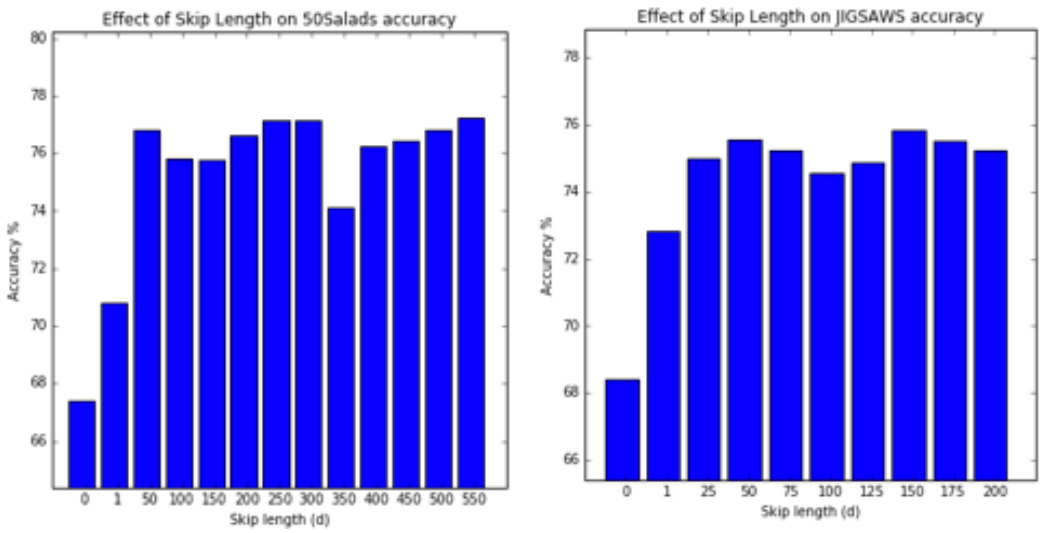


Figure 3.13: Skip Chain length experiments (left) 50 Salads (right) JIGSAWS. The prior is not used in these experiments.

Some experiments were performed using different parameters or test setups so it is important to compare the relative performance within an experiment. Furthermore, there is a small amount of variation in performance on different runs of an experiment with the same model and same hyper-parameters. This is due to the random initialization of some of the weights. On 50 Salads, the standard deviation across 10 trials for the SC-CRF was 0.267%.

At the end, we show graphs of the per-trial performance using our best methods. These graphs highlight that certain users tend to achieve much better or worse performance than others. These graphs can also be used to highlight the complimentary information computed from our metrics.

Skip Chains: Figure 3.13 shows the accuracy on 50 Salads and JIGSAWS using the SC-CRF (Equation 3.1) with the per-frame unary term and skip chain pairwise with

50 Salads

Models	Accuracy	Overlap	Edit
Linear Chain CRF (w/o prior)	71.2	36.9	46.2
Linear Chain CRF (w/ prior)	**73.8**	**43.5**	**52.4**

JIGSAWS

Models	Accuracy	Overlap	Edit
Linear Chain CRF (w/o prior)	73.4	50.0	68.1
Linear Chain CRF (w/ prior)	**78.7**	**57.8**	**74.7**

Table 3.2: Results with and without temporal priors using a Linear Chain CRF, which is the special case of a SC-CRF with $d = 1$.

varying skip lengths. Results are shown for length ranging from $d = 0$ to $d = 500$ for 50 Salads and $d = 200$ for JIGSAWS. Note that the length of a typical action is longer for 50 Salads.

A value of $d = 0$ implies there was no pairwise term and $d = 1$ is the special case of a linear chain CRF. Going from the linear chain to a skip chain increased accuracy by about 5% on 50 Salads and 3% on JIGSAWS. Using skip chains, there was a wide range of lengths that gave similar performance. Once the length gets too big performance starts to drop. For example $d = 500$ achieved 71% on JIGSAWS.

Temporal Prior: Table 3.2 shows the impact of using a temporal prior with a linear chain CRF. Given the limited amount of training data, we set the number of time steps $T_o = 30$ to prevent overfitting. Overlap and edit increase by 5-8% on both datasets. On 50 Salads there was a small improvement in accuracy and on JIGSAWS the increase in performance was larger. As such, the temporal prior is used in all remaining experiments.

50 Salads

Models	Accuracy	Overlap	Edit
SC-CRF	77.45	59.4	51.9
LSC-CRF ($H = 2$)	**79.2**	62.3	**54.3**
LSC-CRF ($H = 3$)	78.8	60.1	52.6
LSC-CRF ($H = 4$)	**79.2**	**62.4**	52.9

JIGSAWS

Models	Accuracy	Overlap	Edit
SC-CRF	79.3	86.0	71.3
LSC-CRF ($H = 2$)	82.1	87.4	**75.2**
LSC-CRF ($H = 3$)	81.4	87.5	74.9
LSC-CRF ($H = 4$)	**82.5**	**88.3**	72.7

Table 3.3: Results on 50 Salads and JIGSAWS using a Latent Skip Chain CRF. $H = h$ defines the number of latent variables. Note that the special case of the LSC-CRF with $H = 1$ is simply the SC-CRF. The prior and filtering are both used in all trials.

Latent states: We looked at the effect of using a latent skip chain CRF (Equation 3.11) with varying numbers of latent states, H, per class. These results are shown in Table 3.3. Adding latent states results in about a 2% improvement for 50 Salads and 3% improvement for JIGSAWS.

Filtering: Table 3.4 shows performance of the SC-CRF without filtering, with a median filter (width = $d/2$), Segmental Viterbi, and with our segmental inference algorithm. We evaluated Segmental Viterbi with and without segment length constraints. The minimum and maximum segment lengths were based on the true min and max durations from the training sets. In contrast to our original hypothesis, segmental inference only had a minor effect on performance. By some metrics, including accuracy, performance actually dropped slightly but on others performance improved.

50 Salads

CSC-CRF inference	Accuracy	Overlap	Edit
Viterbi (w/ skips)	78.8	38.8	49.8
Segmental Viterbi (min/max)	77.8	**39.7**	33.2
K-Segment Viterbi (ours)	78.3	38.6	48.6
Viterbi (w/ skips) + median filter	**79.3**	38.7	**53.4**

JIGSAWS

CSC-CRF inference	Accuracy	Overlap	Edit
Viterbi (w/ skips)	80.7	69.4	60.3
Segmental Viterbi (min/max)	78.2	65.4	63.1
K-Segment Viterbi (ours)	78.6	66.4	66.3
Viterbi (w/ skips) + median filter	**81.3**	**70.4**	**80.4**

Table 3.4: Results on 50 Salads and JIGSAWS using the CSC-CRF using Viterbi inference with and without median filtering and segmental inference using Segmental Viterbi (with duration constraints), and our K-Segment algorithm. For both datasets $d = 100$.

Under most metrics, the median filter with a large window ($w = d/2$) achieved the best performance. Notice that edit scores on 50 Salads were relatively low across all inference methods. This is largely due to the background class, which often occurs before or after some other actions. The median filter is more efficient and performs better in aggregate so we used this kind of filtering for the rest of the experiments.

Action Primitives: Table 3.5 shows performance of the LC-SC-CRF using action primitives for several numbers of latent states per class. Note that CSC-CRF is the special case of the LC-SC-CRF without latent states ($H = 1$). The length of each action primitive is the same for all cases, which is the same as the skip length d. On both datasets accuracy does improve by 1-3% with the use of latent filters, however, performance quickly plateaus as the number of filters per class increases. We did not

50 Salads

Models	Accuracy	Overlap	Edit
SC-CRF	77.5	39.4	49.8
CSC-CRF	81.0	43.4	55.2
LC-SC-CRF ($H = 2$)	80.9	43.6	54.5
LC-SC-CRF ($H = 3$)	**81.3**	44.2	**55.5**
LC-SC-CRF ($H = 4$)	81.1	**44.5**	55.0

JIGSAWS

Models	Accuracy	Overlap	Edit
SC-CRF	79.3	86.0	71.3
CSC-CRF	81.3	70.4	80.4
LC-SC-CRF ($H = 2$)	**82.3**	**72.2**	**81.8**
LC-SC-CRF ($H = 3$)	82.2	72.0	81.1
LC-SC-CRF ($H = 4$)	81.9	71.9	79.7

Table 3.5: Results on 50 Salads and JIGSAWS. $H = h$ defines the number of latent variables. When $H = 1$ this is the special case of the CSC-CRF without latent variables. Inference is performed using Viterbi (w/ skips) with a median filter to smooth out predictions. For JIGSAWS $d = 100$ and for 50 Salads $d = 200$.

find any improvement above $H = 4$ on either dataset.

Granularities: Table 3.7 shows 50 Salads results for each action granularity. We are unaware of any prior results using these granularities. Our model performed very well on high-level actions, which is not very surprising given there are only four action classes. Performance on mid-level actions was lower. In this setup there are 18 classes, such as `cutting_cucumber` and `cutting_cheese`, some of which are indistinguishable using the accelerometer data. Superior performance on this granularity likely requires the aid of computer vision models to recognize each ingredient. Performance on low-level actions was worse, however, given there are 52 action classes we do substantially better than chance (random accuracy = 1.9%). This granularity includes very fine-

50 Salads

Models	Accuracy	Overlap	Edit
DTW	65.4	30.3	**62.0**
SVM	69.1	33.9	17.3
RNN-LSTM [125]	73.3	-	54.5
LC-SC-CRF ($H = 3$)	**81.3**	44.2	**55.5**

JIGSAWS

Models	Accuracy	Overlap	Edit
SVM	67.7	55.0	36.4
MsM-CRF [33]	67.8	-	-
KSVD-SHMM [30]	73.5	-	-
GMM-HMM [31]	74.0	-	-
DTW	75.7	**60.9**	**85.9**
SD-SDL [126]	78.6	-	83.3
LSTM [16]	80.5	-	75.3
LC-SC-CRF ($H = 2$)	82.3	72.2	81.8
Bidir LSTM [16]	**83.3**	-	81.1

Table 3.6: Comparisons with prior work.

grained actions such as `start_cutting_cucumber` and `stop_cutting_cucumber`.

Error analysis: Figure 3.14 shows the accuracy and edit scores for each trial in 50 Salads and JIGSAWS, where each dot represents one trial. Note that the pair of edit and accuracy scores for each column represents the same trial. All trials are sorted in ascending order according to their edit scores. These scores were computed using the LC-SC-CRF ($H = 3$, $d = 100$ for JIGSAWS and $d = 200$ for 50 Salads).

There are several things to take away from these graphs. First, there is little correlation between the accuracy and edit score. We computed a Pearson correlation for the two metrics [127] which has a range of -1 to $+1$. On 50 Salads the ρ value is 0.37 and on JIGSAWS it is 0.73; both of which are weak correlations. Visually, it is

Low-level actions	Accuracy	Overlap	Edit
SC-CRF	44.04	27.69	26.0
CSC-CRF	44.76	32.17	29.45
LC-SC-CRF ($H = 3$)	**46.28**	**34.3**	**31.71**
Mid-level actions	Accuracy	Overlap	Edit
SC-CRF	51.47	32.98	20.62
CSC-CRF	52.36	34.4	26.33
LC-SC-CRF ($H = 3$)	**55.05**	**38.42**	**29.02**
Highest-level actions	Accuracy	Overlap	Edit
SC-CRF	92.85	60.86	57.9
CSC-CRF	93.26	59.59	**64.27**
LC-SC-CRF ($H = 3$)	**94.06**	**64.64**	63.24

Table 3.7: Evaluation on the 50 Salads dataset using the low-level, mid-level, or highest-level actions described in the text.

Labels	**Duration**	**# Segs**	**Speedup**
50 Salads (Low)	2289	65	35x
50 Salads (Mid)	3100	25	124x
50 Salads (Higher)	3100	24	129x
50 Salads (Highest)	11423	6	1902x
JIGSAWS	1107	37	30x

Table 3.8: Speedup Analysis of our segmental inference algorithm compared to Segmental Viterbi.

clear that an increase in edit score does not necessarily correlate with an increase in accuracy. For example, trial 8 on 50 Salads has an accuracy/edit of about 69%/41%, trial 9 has 88%/43%, and trial 10 has 73%/44%. The correlation on JIGSAWS is a little bit higher, but there are still large oscillations in accuracy for marginal changes in edit.

There are five trials in JIGSAWS for which we achieve 100% edit. This is perfect for applications in which we only care about the ordering of actions. It is interesting,

however, to see the accuracy of these five predictions only reaches about 90%. In this case, all of the errors come from temporal shifts in the boundaries between actions.

Informally we have found that our models achieve much worse performance with certain users. For example, in JIGSAWS user "G" makes several errors, which results in a very different sequencing of actions than any of the other users. As such, the accuracy and edit scores for this user are much lower. The first three trials in Figure 3.14 (right) correspond to this user.

On both datasets the accuracy saturates around 90%. Due to limitations in the annotations for both datasets we do not know whether we can expect higher performance. Ideally, we could compare this number to the inter-annotator variability, however, the datasets we collected only have one set of labels[5]. By comparing multiple label sets we could estimate the best potential performance.

As a reference point, in surgical skill applications inter-annotator agreement when deciding on whether a user is a novice or an expert is relatively low (e.g., 75% evaluating on surgical training tasks in [128]). In our case, there is likely large variation in when each action starts and stops. On the M2CAI2016 surgical workflow dataset in Chapter 6 we take these temporal shifts into account by adding a buffer around each action transition.

Speedup: Table 3.8 shows the speedup of our K-segment inference algorithm compared to Segmental Viterbi on all 50 Salads and JIGSAWS label sets. One practical

[5]In each dataset there were multiple people verifying the labels, but only one set of "true" labels.

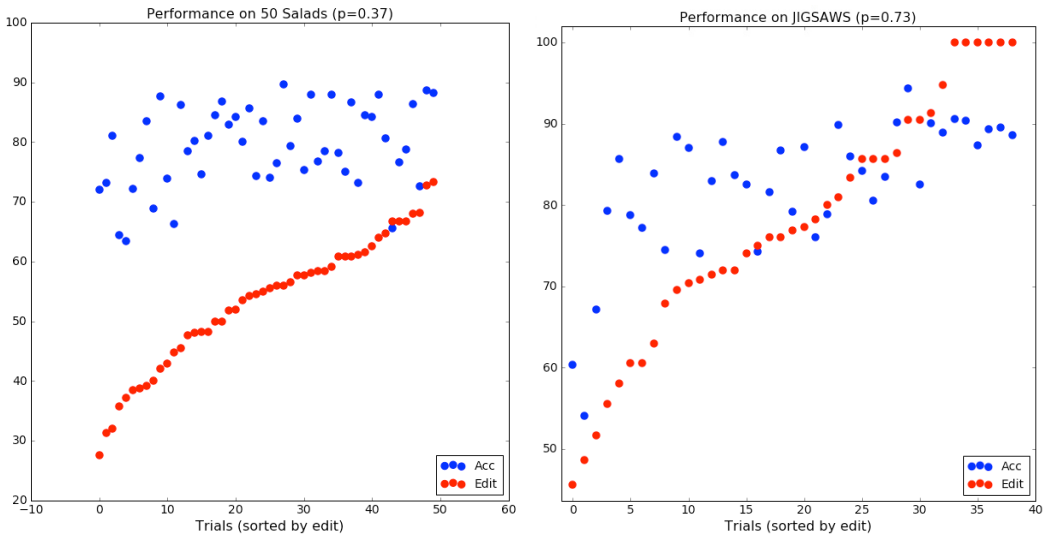


Figure 3.14: Accuracy and edit scores for (left) 50 Salads (right) JIGSAWS. Trials are sorted by edit scores in ascending order. Results are from evaluation with the LC-SC-CRF with d=100 (JIGSAWS) and d=200 (50 Salads) with 3 latent states per class. "p" in the title of each refers to the Pearson ρ correlation.

implication is that our algorithm scales readily to full-length videos. On 50 Salads, Segmental Viterbi took 2 hours to compute highest-level predictions for all trials compared to a mere 4 seconds using ours.

Other experiments: Throughout this research we assessed several methods for independently segmenting actions and then classifying them. In our experience there is no single temporal segmentation method that works well on all kinds of time series data. One method may work well on 50 Salads but poorly on JIGSAWS. Furthermore, by performing these tasks independently it is common to grossly over- or under-segment the data. It is harder to predict the correct action sequence in both of these cases.

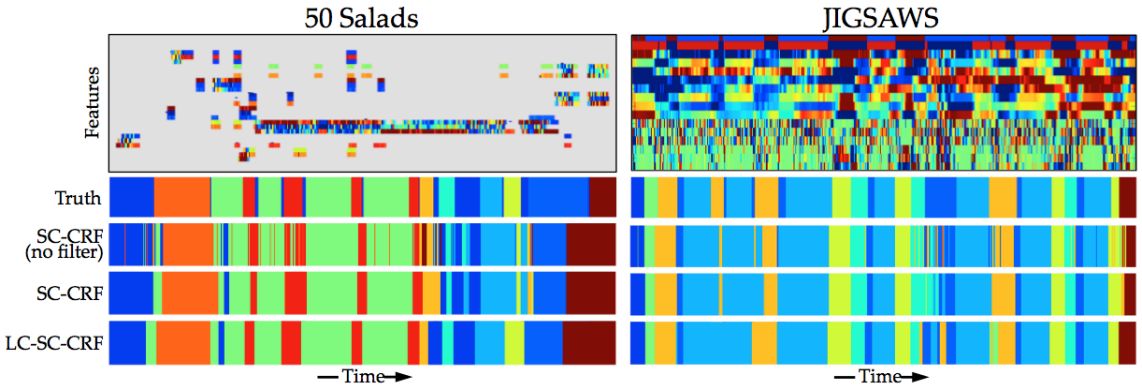


Figure 3.15: Examples from 50 Salads and JIGSAWS. The top images show sensor signals over time where red is high, green is neutral, and blue is low. On 50 Salads gray denotes zero-acceleration. Subsequent rows depict the ground truth and predicted labels for several models ($H = 1$). Each color corresponds to a different action class.

3.6 Visualization

Figure 3.15 shows example sensor data and predictions for each dataset. The top depicts sensor signals throughout a sequence. Red is high, green is neutral, and blue is low. On 50 Salads gray corresponds to zero acceleration. Consecutive plots show ground truth and predicted action sequences for several versions of our model. Each color indicates a unique action. The model without filtering clearly contains many incorrect frames. With the LC-SC-CRF segment boundaries tend to be better aligned with the ground truth and there are fewer over-segmentation issues than with the SC-CRF.

There are many instances in 50 Salads where the signals do not appear to be aligned exactly with the accelerometer values. While it looks like there is a synchronization issue, this actually shows a limitation in using accelerometer features for

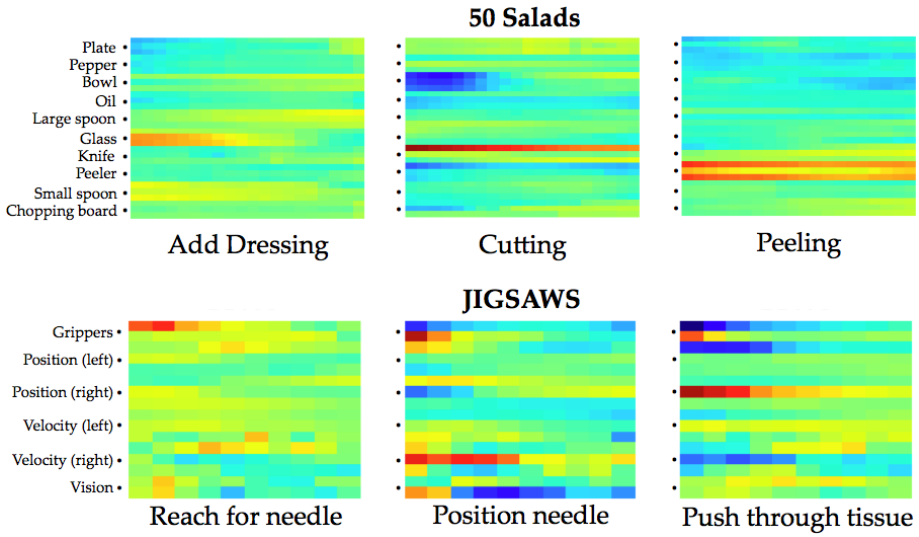


Figure 3.16: Example action primitives learned on 50 Salads and JIGSAWS. Each row in an image depicts the weights for that corresponding sensor over time. In 50 Salads there are X, Y, and Z values for each object.

action recognition. In many cases, an action starts when the user first moves her or his hand to the cooking utensil, as opposed to starting when the utensil has physically moved. This happens in actions like `peeling` and `cutting`. If we extracted information from the video it is possible that we would pick these events up earlier.

On both datasets our action primitives provide an interpretable way for learning how signals transition throughout an action. Figure 3.16 highlights some example action primitives. In 50 Salads each object has signals for the X, Y, and Z components of each accelerometer. Typically there is one dominant object that corresponds to each action class. For example in *cutting* the knife is dominant and in *peeling* the peeler is dominant. Notice that other objects sometimes vary in shades of blue and green. These may be used for portions of the task (e.g. the start) or only used in some instances of an action.

In JIGSAW it is common for an action to contain a change in gripper state. For example when reaching for the needle the user starts with the gripper open (red) and then closes it on the needle (green). Other actions are often characterized by transitions in tool positions. See the gradients in the "Position (right)" features in `position_needle` and `push_through_tissue`.

Some action primitives are more interpretable than others. The clarity appears to be a function of the sensor type. In 50 Salads, when a user changes utensils there is a clear delineation between the object being in-use versus not in-use. In JIGSAWS the action primitives are more blurred which is due to the continuous positions and velocities in the robot kinematics data.

3.7 Conclusion

In this chapter we described a model for sensor-based action segmentation that outperforms several baselines and competing methods on JIGSAWS and 50 Salads. The first key component, the skip chain pairwise term, exploited the disconnect between the sensor sample rate and the rate of actions to better capture transitions between action labels. The second key component, latent action primitives, captured temporal correlations between sensor signals as a function of each action class. These are relatively efficient computationally and produced visually interpretable convolutional filters. Our third technical contribution was an efficient constrained segmental inference algorithm. While it did not perform as well as simpler filtering methods,

it is worth further exploration. Future work should explore the use of these three mechanisms in RNN-based models.

Our models and learning methods from this chapter were implemented from scratch in Python using Numba, an LLVM-based Just In Time compiler. The code, including scripts for evaluating on JIGSAWS and 50 Salads, is available online[6].

In the next chapter we will introduce a video-based feature representation that will be used in tandem with the segmental and LC-SC-CRF models.

3.8 Appendix: Connections to RNNs

There has been significant recent interest in Recurrent Neural Networks (RNNs) for time series analysis. There are many variants on RNNs, but the vanilla model has a similar structure to a latent linear chain CRF, as shown in Figure 3.17. Note that there are two typical depictions of an RNN: the first is as a simple feed forward neural network with a recurrent edge and the second is an unrolled graph as shown here. Both are mathematically equivalent. We leave out recent work using memory networks like Long Short Term Memory (LSTM) [129, 130] and Gated Recurrent Units (GRUs) [131]. For a recent introduction to RNNs see [99].

[6]Code: `https://github.com/colincsl/LCTM/`

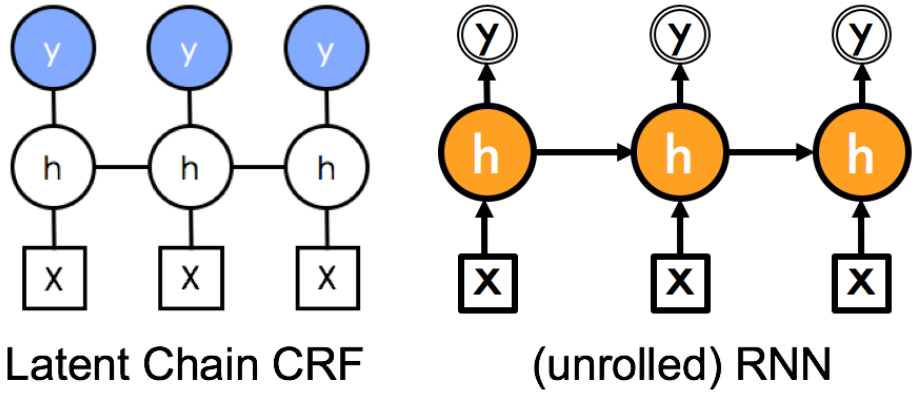


Figure 3.17: (Left) Latent Linear Chain Conditional Random Field and (right) An unrolled vanilla Recurrent Neural Network.

RNN Formulation

A (vanilla) RNN is defined using input data vectors $X_t \in \mathbf{R}^F$, hidden state vectors $h_t \in \mathbf{R}^H$, and predicted probabilities $y_t \in \mathbf{R}^C$ for all times t from 1 to T where F is the number of input features, H is the number of (shared) hidden states, and C is the number of classes. There are weight matrices $W_x \in \mathbf{R}^{H \times F}$, $W_h \in \mathbf{R}^{H \times H}$, and $W_y \in \mathbf{R}^{C \times H}$ and corresponding bias vectors $b_h \in \mathbf{R}^H$, and $b_y \in \mathbf{R}^C$ which are shared across time. The hidden state is

$$h_t = \tanh(W_x X_t + W_h h_{t-1} + b_h) \tag{3.35}$$

and the class probability is estimated using the softmax function

$$y_t = p(y_t | x_1, \ldots, x_t) = \text{softmax}(W_y h_t + b_y), \tag{3.36}$$

where a softmax is defined as

$$\textbf{softmax}(x_c) = \frac{\exp(x_c)}{\sum_{c'=1}^{C} \exp(x_{c'})} \tag{3.37}$$

for arbitrary vector $x \in \mathbf{R}^C$ [110].

The value of y_t is only a function of the hidden state at the time step but the value of h_t depends on the data at t and the hidden states from previous time steps. Note that this formulation is causal, meaning that the prediction at t is only a function of the data from 1 to t. We discuss Bi-direction RNNs in Chapter 5.

Connections between Latent Chain CRFs and RNNs

Latent terms: In both cases, the prediction at each time step is a function of the hidden nodes at that time. In our CRF, we defined a set of class-specific hidden states where only one is active for any given time step. RNNs assume that there are a set of class-agnostic hidden states and for any time step the prediction is a linear combination of the values for all hidden states.

Inference: All labels in a chain CRF are predicted jointly by applying the Viterbi algorithm on the whole sequence. RNNs are typically computed in a feed forward manner, and thus the label at each time step is simply a function of the hidden states at that time step.

Skip Chains: For computational reasons, we only used one skip duration. Adding more edges quickly becomes intractable using exact inference and is computationally

equivalent to an nth-order CRF [71]. Inference in an RNN is performed per-frame and thus adding multiple edges has a negligible effect on the cost of inference. In fact, the recent Clockwork RNN model uses multiple skip-like edges [100].

Training: RNNs are typically trained using Stochastic Gradient Descent (SGD) with a categorical cross-entropy loss function. Back Propagation Through Time (BPTT) [132] is used to unroll the network and compute the gradients for each parameter. CRFs can be trained using probabilistic or max-margin approaches. In the discriminative case we use SGD with the Structural SVM loss function. The gradients are computed using loss-augmented inference. It would be interesting to explore the use of the SSVM loss with an RNN.

Action Primitives: Given that our action primitives are linear functions of filters and a window of input data, it is reasonable to replace the typical data term in an RNN with our action primitive term. This would result in one (shared) hidden state per primitive.

We leave RNN variants of our skip-chain and action primitive terms as future work.

Chapter 4

Spatiotemporal Models and Sensor Substitution

State-of-the-art video-based action segmentation models typically fail to achieve a level of performance near that of domain-specific sensors. In this chapter we introduce video-only and multi-modal spatiotemporal models that work towards minimizing this performance gap.

4.1 Introduction

Domain-specific sensors are often critically important in robotics applications. For example, pressure sensors are important for manipulation tasks, tool tracking supports visual servoing, and accelerometers support SLAM approaches. However, there

are many applications where the ideal sensing is too impractical or too costly to deploy in real-world settings. For instance, in the next-generation kitchen[1] it might be beneficial to attach motion sensors to all tools to support monitoring or control, but retrofitting every kitchen at scale is impractical due to a combination of cost, data acquisition and synchronization overhead, and physical constraints in instrumentation. A practical alternative would be to mount a video camera to observe the scene, but current methods for video tracking and action recognition generally achieve worse performance than their counterparts using domain-specific sensing (e.g.,[33, 112]).

State-of-the-art approaches for fine-grained action segmentation, as described in Section 2, use bag of words models with hand-crafted features such as Improved Dense Trajectories [9]. These features only capture very low level texture patterns and do not capture spatial relationships between objects. In this chapter, we develop video-only and multi-modal Spatiotemporal Convolutional Neural Networks (ST-CNN). Our models factorize video into a spatial component that hierarchically captures texture patterns in different regions of an image and a temporal component that captures how the patterns change over time. The temporal component extends the temporal action primitives from Chapter 3 to video-based spatiotemporal primitives. Our approach achieves significantly higher performance than a Dense Trajectory baseline and a VGG-based [95] spatial CNN baseline. Performance using our video approach – but does not outperform – the sensor-based results and is closer than the state of

[1] `http://www.conceptkitchen2025.com/`

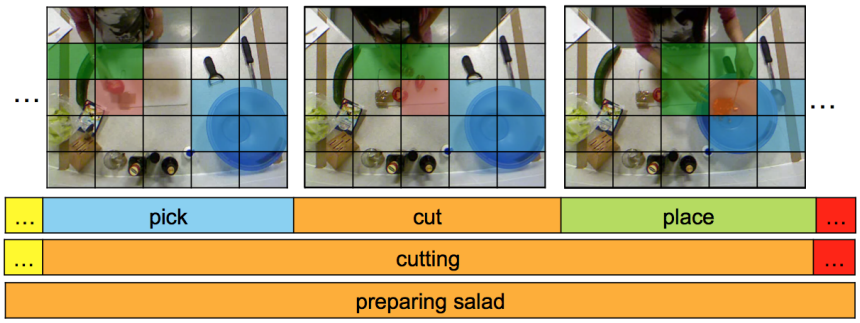


Figure 4.1: Our model captures object relationships and how these relationships change temporally. (top) Latent *hand* and *tomato* regions are highlighted in different colors on images from the 50 Salads dataset. (bottom) We evaluate on multiple label granularities that model fine-grained or coarse-grained actions.

the art for video.

For concreteness, throughout this chapter we refer to the sub-sequence depicted in Figure 4.1: A user places a tomato onto a cutting board, cuts it with a knife, and places it into a salad bowl. This is part of a much longer salad preparation sequence. The spatial component should capture the location of the tomato with respect to the cutting board and bowl, and the temporal component should capture the tomato's transition from being on the cutting board at the start of an action to being in the bowl at the end.

Many situated tasks, including surveillance and surgical training assessment, employ a fixed camera to monitor human actions. We leverage this fixed-camera assumption to investigate the ability of our model to capture geometric relationships between objects. This is encoded in the spatial component of our model, in which we use a simplified variation of the VGG [95] CNN architecture. This model can be viewed as having two parts. In the first half of the network a hierarchy of convolu-

tional filters are applied to capture local texture patterns within different regions of a given image. For example, these filters may encode a tomato or bowl texture. In the second half of the spatial network a set of fully connected layers capture correlations between the texture patterns and their spatial locations. We visualize the ability of our network to successfully capture this information later in the chapter.

For some applications, fine-tuning a CNN that was originally trained on a large image classification dataset (e.g., ImageNet) can yield high performance on a new task (e.g., [54]). Unfortunately, we show in the results section that this is not true for action segmentation in situated environments. In our first set of results we train a network solely using a given dataset and show that it achieves superior performance than a model trained on ImageNet. We note that this might not be practical for many applications where is it too costly to hand-label copious amounts of training data. As such, we describe a notion of *sensor substitution*; instead of training a spatial CNN with manually-annotated action labels we train it using sensor values such as robot kinematics or accelerometer values. A classifier is then learned on top of this network to predict which action is occurring. We show that not only can we use this to predict actions, but the network can generate *virtual* sensor signals with sufficient performance, which enables us to substitute video for sensors in tasks beyond action segmentation. Our sensor substitution model achieves very high performance when training and evaluating on the same surgical training task (e.g., Suturing) and good performance when evaluated on other surgical training tasks (e.g., Knot Tying). We

highlight the ability of our model to capture this sensor information and to generate virtual sensor signals through multiple visualization techniques.

We combine our spatial networks with a variation of our convolutional action primitives that captures how object relationships change over the course of an action. These temporal primitives are learned using the latent features output by the spatial CNN as input. These filters are on the order of 10 seconds long and explicitly capture mid-range motion patterns. The primary difference between our previous action primitives and the new ones, is that we now take linear combinations of shared filters that are common between all actions as opposed to the maximum over class-specific filters. The probability of an action at any given time is estimated using 1D convolutions over the spatial activations. On both datasets we achieve notably higher results than our baseline video models.

In summary our contributions are: (1) We develop a Spatiotemporal CNN which captures information about object relationships and how relationships change over time, (2) we exploit sensor data during training time to learn better video-based models that can predict domain-specific sensor signals, (3) when combined with the segmental and LC-SC-CRF models from Chapter 3, as depicted in Figure 4.1, we substantially outperform recent methods for fine-grained recognition on two challenging datasets.

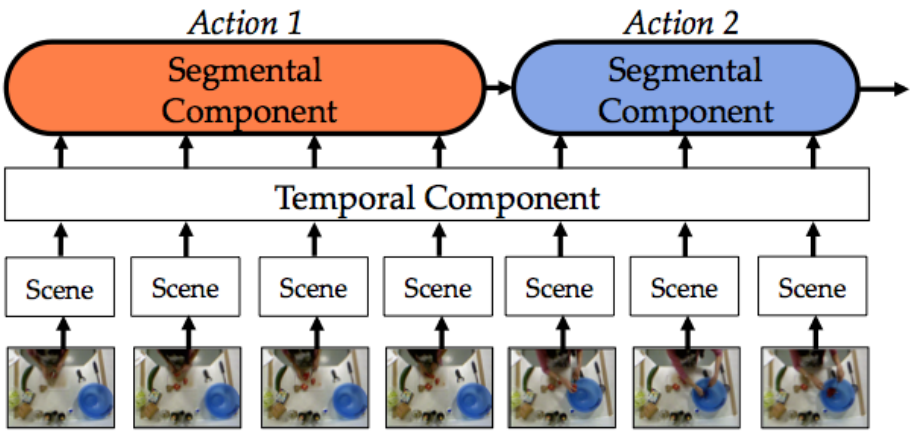


Figure 4.2: Our model contains three components. The spatial, temporal, and segmental units capture object relationships, how those relationships change, and how actions transition from one to another, respectively.

4.2 Spatial Component

In this section, we introduce a CNN topology inspired by VGG [95] that uses hierarchical convolutional filters to capture object texture and spatial location. We introduce the network architecture, as depicted in Figure 4.3, and highlight differences between our approach and other CNNs. Later we describe our notion of sensor substitution. The input into our spatial CNN is a set of color and motion images for each frame. The output is a feature representation at every frame that describes the content of the scene. For a recent introduction to CNNs see [110].

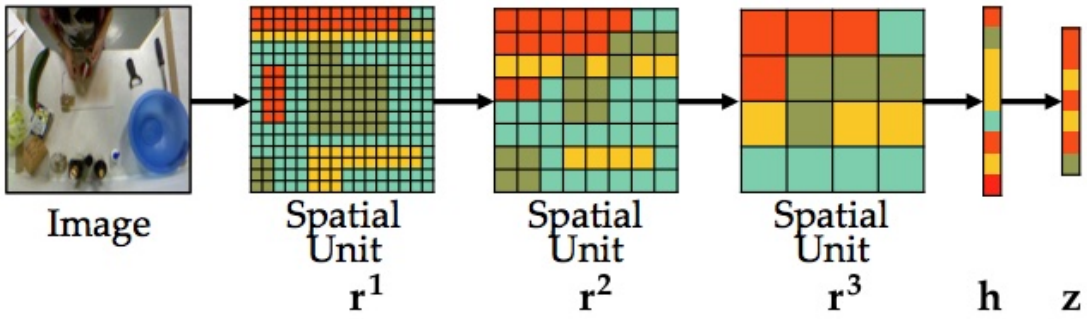


Figure 4.3: The spatial component is a CNN modeled specifically to capture latent object relationships.

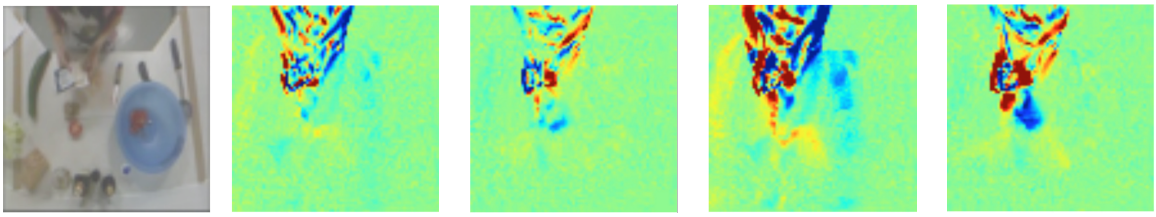
4.2.1 CNN Input

For each time t there is an image pair $I_t = \{I_t^c, I_t^m\}$, where I_t^c is a color image and I_t^m is a motion image. The motion image captures when an object has moved into or out of a region and is computed by taking the difference between frames across a short time interval. Let $\delta(\cdot)$ be our image difference function such that

$$\delta(I^c, t, d) = \frac{1}{3}\sum_{i\in\{r,g,b\}} I_t^{c,i} - \frac{1}{3}\sum_{i\in\{r,g,b\}} I_{t-d}^{c,r} \tag{4.1}$$

where d is a temporal offset and i corresponds to the RGB color channels. This is effectively computing a gray scale image for each time step and taking the difference.[2] Unless otherwise specified we set $I_t^m = [\delta(I^c, t, 2d), \delta(I^c, t, d), \delta(I^c, t, -d), \delta(I^c, t, -2d)]$ which is a set of difference images for two steps forward and two steps backward. For

[2]We investigated several variation on this, including defining a separate difference image for each of the RGB channels, but ultimately found that the change in performance was negligible and that separating the channels required more GPU memory. We also compared against a version using the absolute value of the differences. In preliminary experiments we found that the version in the text, which can have both positive and negative values, achieved superior performance.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 4.4: The input into the spatial CNN is an RGB image I_t^c (a) and a set of difference images I_t^m (b - e). Red indicates a high positive difference, blue indicates a high negative difference, and green indicates no difference.

50 Salads this offset is $d = 0.5$ second. The areas with high difference values tend to be relevant to the ongoing action, so this function can be viewed as a simple type of attention mechanism. An example image set from 50 Salads is shown in Figure 4.4.

Note that in our first paper using this network [112] we used Motion History Images [133] as a motion image. When properly tuned these work well, but our simple image differencing variant works better on a larger variety of datasets. Other work (e.g. [62]) has shown success using optical flow as a motion image. In preliminary experiments, we found that optical flow did not sufficiently capturing small hand motions and was too noisy due to video compression.

4.2.2 Spatial CNN

Our spatial model is a variation on the VGG CNN, as shown in Figure 4.3, and can be described using a set of L spatial units. Each spatial unit is composed of a convolutional layer with F_l filters of size 3×3, a Rectified Linear Unit (ReLU), and 3×3 max pooling. If the resolution of our input is $N_0 \times N_0$, the output resolution

after one spatial unit is $N_1 \times N_1$ where $N_1 = \frac{N_0}{3}$. For each of the $R_l = N_l^2$ regions in the l-th layer, there is a set of activation vectors $r^l = \{r_i^l\}_{i=1}^{R_l}$, where $r_i^l \in \mathbb{R}^{F_l}$, that encode the texture in that region on the image. These regions, and corresponding feature vectors, are depicted by the colored blocks in Figure 4.3.

The output of the L-th spatial unit, r^L, is fed into a fully connected layer which has F_{fc} states that capture relationships between regions and their corresponding latent object representations. For example, a state may produce a high score for *tomato* in the region with the cutting board and *knife* in the region next to it. The state $h \in \mathbb{R}^{F_{fc}}$ is a function of weights $W^{(0)} \in \mathbb{R}^{F_{fc} \times F_L \cdot R_L}$ and biases $b^{(0)} \in \mathbb{R}^{F_{fc}}$:[3]

$$h = \text{ReLU}(W^{(0)} r^L + b^{(0)}). \tag{4.2}$$

As with other CNNs, we use the Rectified Linear Unit non-linearity $\text{ReLU}(x) = \max(0, x)$.

The above spatial component, when applied to frame t, produces state vector h_t. We train the spatial component with auxiliary labels $z_t \in \{0, 1\}^C$, where C is the number of classes, denoting the ground truth action label for each time step. We predict the probability, $\hat{z}_t \in [0, 1]^C$, of each action class at that frame using the

[3]For notational clarity we denote all weight matrices as $W^{(\cdot)}$ and bias vectors $b^{(\cdot)}$ to reduce the number of variables.

softmax function:

$$\hat{z}_t = \text{softmax}(W^{(1)} h_t + b^{(1)}), \tag{4.3}$$

where $W^{(1)} \in \mathbb{R}^{C \times F_{fc}}$ and $b^{(1)} \in \mathbb{R}^C$. The softmax function is defined as

$$\textbf{softmax}(x_c) = \frac{\exp(x_c)}{\sum_{c'=1}^{C} \exp(x_{c'})} \tag{4.4}$$

for arbitrary vector $x \in \mathbf{R}^C$ [110]. This function outputs a vector corresponding to the per-class probabilities under the categorical distribution.

Note, $\hat{z}_t$ is computed solely for the purpose of training the spatial component. The input to the temporal component, described later, is h_t.

We visualize this network in several different ways later, but in efforts to instill an intuition for what this model tends to capture, we include an example visualization here. Figure 4.5 shows example CNN activations after each spatial unit. The top row shows the sum of all filter activations after the l-th layer and the bottom row shows the color corresponding to the best scoring filter at each pixel location. Notice the relevant objects in the image and the regions corresponding to the action all have high activations and different best-scoring filters. This implies that the learned filters are similar to mid-level object detectors. We expand on this in Section 4.6.

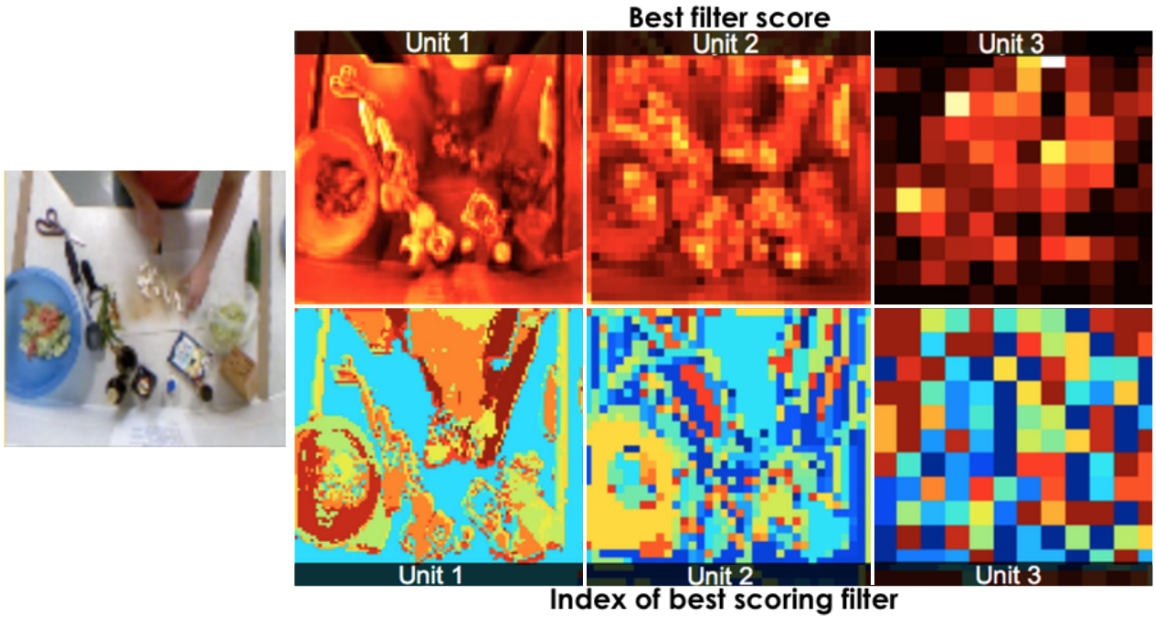


Figure 4.5: The user is moving cucumber slices from the cutting board to the salad bowl. The top-right images show the sum of all feature activations after each spatial unit from the scene CNN. Notice that the cutting board and bowl regions have the highest (yellow) activations whereas the unimportant regions have low (black and red) activations. In the bottom images, colored boxes correspond to the filter index with the highest activation in that region. Different objects tend to be activated with different filters.

4.2.2.1 Relationships to other CNNs

Our CNN is inspired by deep networks for image classification such as VGG and AlexNet but differs in ways that are important for fine-grained action recognition. In particular, these differences are due to the fact that we assume the camera in each dataset is fixed – which enables us to study the network's ability to capture geometric relationships between objects – and that we want to capture subtle differences between actions as opposed to classifying diverse scenes or objects.

Similar to VGG, we employ a sequence of spatial units with common parameters

like filter size. However, in contrast to VGG, which uses two consecutive convolution layers in each spatial unit, we found that the second convolutional layer had negligible impact on performance. Normalization layers, like in AlexNet, did not improve performance either. Overall our network is shallower, has fewer spatial regions, and contains only one intermediate fully connected layer between the convolutional filters and the output. In addition, excessive data-augmentation, including large image rotations and translations, introduced unwanted spatial and rotational invariances, which had a negative impact on our performance. We did find that a small amount of augmentation improved performance in many cases. We randomly rotate each image by up to $\pm 5\%$ and translate each image in the X and Y direction by up to 16 pixels (15% of the image size).

We performed cross validation – training on all except one split and testing on the held out split – using one to seven spatial units and grid sizes from 1×1 to 9×9. We found three spatial units with a 3×3 grid achieved the best results. By contrast, deep networks for image classification tend to use at least four spatial units and have larger grid counts. For example, VGG uses a 7×7 grid and AlexNet uses a 12×12 grid. They use larger sizes because there is a lot more geometric (and textural) variation in the datasets they are applied to. A low spatial resolution naturally induces more spatial invariance. This invariance is a function of the pooling operation; lower resolution implies that information is pooled over a larger portion of the image. We use a low spatial resolution to prevent overfitting on our relatively small datasets. More training

data may be necessary to capture all object configurations if the grid resolution is larger.

One important benefit to our smaller network is that it can be trained in much less time, because there are many fewer layers and filter counts, and it takes less memory. Storing all of the activations for one image using VGG requires 96 Mb of memory whereas our network only requires 4 Mb.

4.2.2.2 Implementation details

We learn parameters $W = \{W^0, W^1\}$, $b = \{b^0, b^1\}$, and the convolutional filters with the categorical cross entropy loss function, which is commonly used in image classification networks such as AlexNet and VGG. A cross entropy loss for each frame t is defined as

$$\mathcal{L}_t = -\sum_{c \in \mathcal{Y}} p(c) \log q(c) \tag{4.5}$$

for probability distributions p and q [110]. p refers to the true distribution, as given by the ground truth labels, and q refers to the probabilities estimated by our network. Given the true label for a training sample, $p(c) = 1$ for the correct class and $p(c) = 0$ for all others. $q(c)$ is given by the estimated probabilities $\hat{z}_t$ from our CNN, which are the output of a softmax function, and thus correspond to values from the categorical distribution. Only one class in the summation has a non-zero value, so the loss

simplifies to

$$\mathcal{L}_t = -\log(\hat{z}_{t,y_t}). \tag{4.6}$$

The total loss is give by the sum of $\mathcal{L}_t$ for all time steps in all training videos.

Note, it is also common to use the notation $\mathcal{L} = -x_c + \log \sum_{c'=1}^{C} \exp(x_{c'})$ for arbitrary vector $x \in \mathbf{R}^C$ and true class c. This, however, assumes that x is the (unnormalized) score (e.g., $W^{(1)}h_t + b^{(1)}$ in our network).

We minimize this loss with back propagation using Stochastic Gradient Descent with ADAM step updates [134]. Dropout regularization is used on the intermediate fully connected layers. Parameters such as grid size, number of filters, and non-linearity functions are determined from cross validation using one of the splits from each dataset described later. Interestingly, we found that these hyper parameters tend to be very similar between all of our datasets. As such, for all datasets we use $F = \{64, 96, 128\}$ filters in the three corresponding spatial units and $F_{fc} = 128$ fully connected states. We used Keras [135], a library of deep learning tools, to implement our model.

Note that training of this model is much faster than when training deeper models such as VGG or ResNet [136]. A single cross validation fold trained for 15 epochs on JIGSAWS takes about 15 minutes on a Nvidia GTX Titan X, compared to over an hour to train VGG for the same number of iterations.

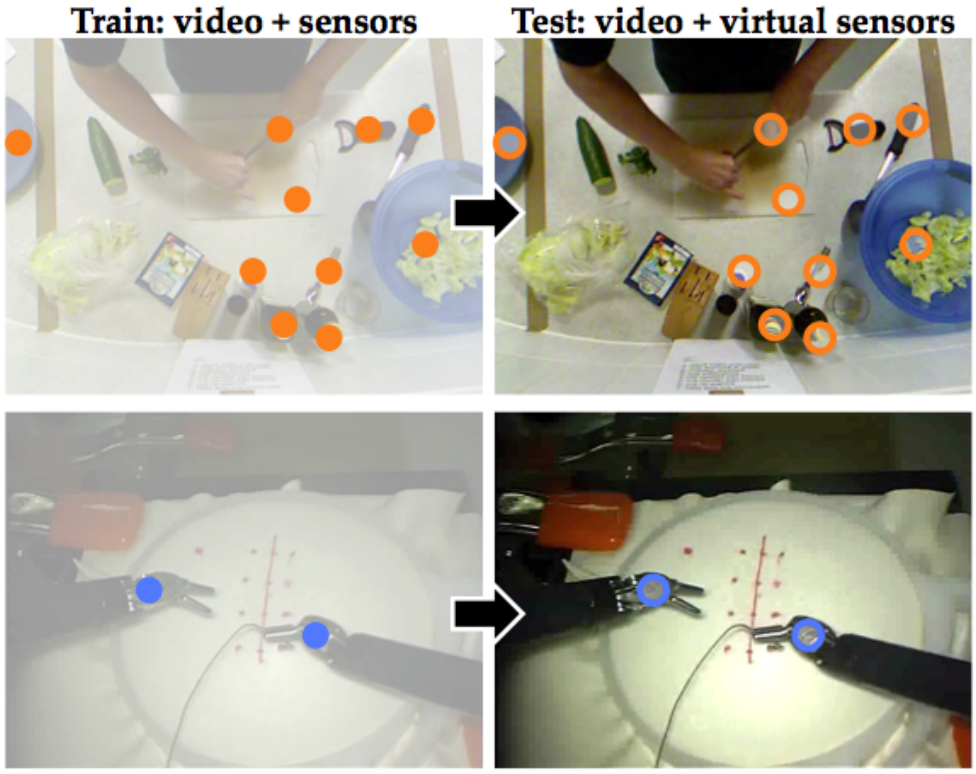


Figure 4.6: Our model learns a video-based representation of domain-specific sensors. (top) Orange circles correspond to accelerometers placed on objects in 50 Salads. (bottom) Blue circles correspond to end effector positions from a da Vinci robot in JIGSAWS. We train on video and sensor data but test on only video by predicting the sensor data from it using this method.

4.2.3 Sensor Substitution

In this section we describe *sensor substitution*, which improves on our spatial CNN in two important ways. First, we show that we can predict sensor data from video with reasonable performance, and second, we show that we can use the regressed "virtual" sensor signals in tandem with our action model from Chapter 3 to achieve high action segmentation performance. One of the primary benefits from this method is that it does not require hand-labeled annotations to learn a spatial CNN. Figure 4.6 depicts

our approach, in which we train with both sensing modalities but evaluate solely using video.

While our methods may apply to a broad range of sensor types (e.g., force sensors, pressure sensors), in this chapter we use two specific types: position data and accelerometer data. For the first type, we regress position measurements by learning a continuous model for the 3D position of two robot end effectors on a da Vinci surgical robot. For the second type, we regress accelerometer signals for each of the ten kitchen tools in the 50 Salads dataset with the goal of identifying when specific objects in a scene are in motion. Note that there is a semantic difference between an acceleration of zero and an acceleration greater than zero, so in our evaluation we assess the ability to detect if each object is `in motion` or is `not in motion`. Note that while the accelerometers capture acceleration – and do not directly capture velocity – we use these interchangeably. It is rare for an object to have non-zero velocity and zero acceleration. This is apparent looking at Figure 4.7, which displays the magnitude of each accelerometer for two trials in the dataset.

There has been related work regressing positions for task such as human pose estimation [137, 138, 139], where CNNs are used to regress 2D or 3D body joint positions given a region of interest an image around a human. In general object detection, recent methods like Faster R-CNN [140] and YOLO [141] use regression CNNs to detect multiple objects in an image.[4] However, we are not aware of any

[4]We are abusing terminology when we differentiate classification CNNs and regression CNNs – all CNNs are technically performing regression. The purpose is to differentiate cases in which the

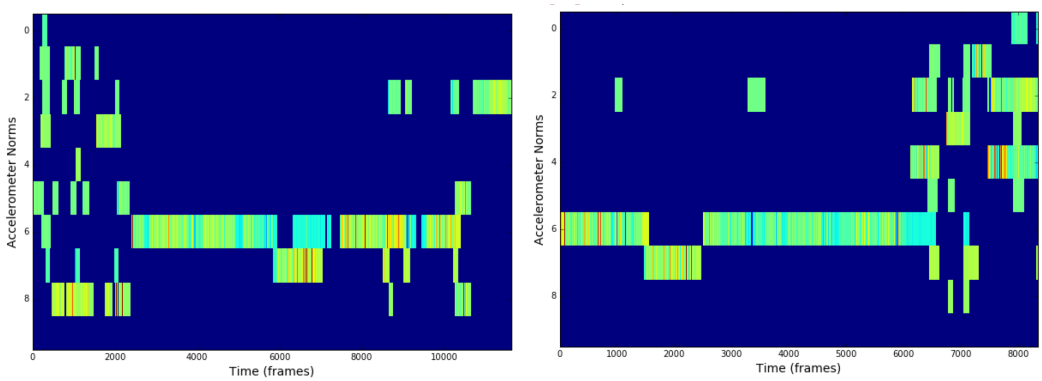


Figure 4.7: Each image visualizes the magnitude of each accelerometer for a trial in 50 Salads.

work that regresses sensor data from video using CNNs.

Model

As with our previous spatial model, the CNN is defined using a sequence of convolutions, ReLUs, and pooling. We use the same architecture as shown in Figure 4.3 except we modify the final output equation using $\hat{z}_t^s \in \mathbf{R}^{F_s}$, where F_s is the number of sensor signals. The output is given by

$$\hat{z}_t^s = W^{(1)} h_t + b^{(1)}. \tag{4.7}$$

This is similar to Equation 4.3 except $W^{(1)} \in \mathbb{R}^{C \times F_S}$ and $b^{(1)} \in \mathbb{R}^C$. The true sensor signals are given by z_t^s for each time t. By contrast, in the previous setup we were predicting the probability of a frame belonging to each class, whereas in this setup we are predicting a real-valued output for each component of the sensor signal.

final output is discrete versus continuous. This is common in the literature.

Instead of using the cross entropy loss, which is computed using the probabilities of a discrete set of outputs, we use the mean squared error loss:

$$\mathcal{L}_t^s = \|z_t^s - \hat{z}_t^s\|_2^2 \tag{4.8}$$

for each time step t. We performed experiments using Tukey's biweight loss as used by Belagiannis *et al.* [137], who claim the biweight loss is better at regression tasks because it is less susceptible to outliers in hand-labeled training data. However, in preliminary experiments we found that using an L2 training objective achieved superior performance. This may be because our sensor measurements do not suffer from the same types of noise as the hand-labeled position data in [137].

We normalize the sensor data to be zero mean with unit standard deviation during training time and apply the inverse for evaluation. We train the network with a batch size of 64 samples and again use Stochastic Gradient Descent with Adam step updates.

4.3 Temporal Component

The models described so far only capture information from a given frame and its corresponding difference images. In most situations, it is unreasonable to expect our model can accurately identify an action based on an image at a single time point. We introduce a temporal component that explicitly captures how the scene changes over the course of an action. We learn a set of temporal convolutional filters that

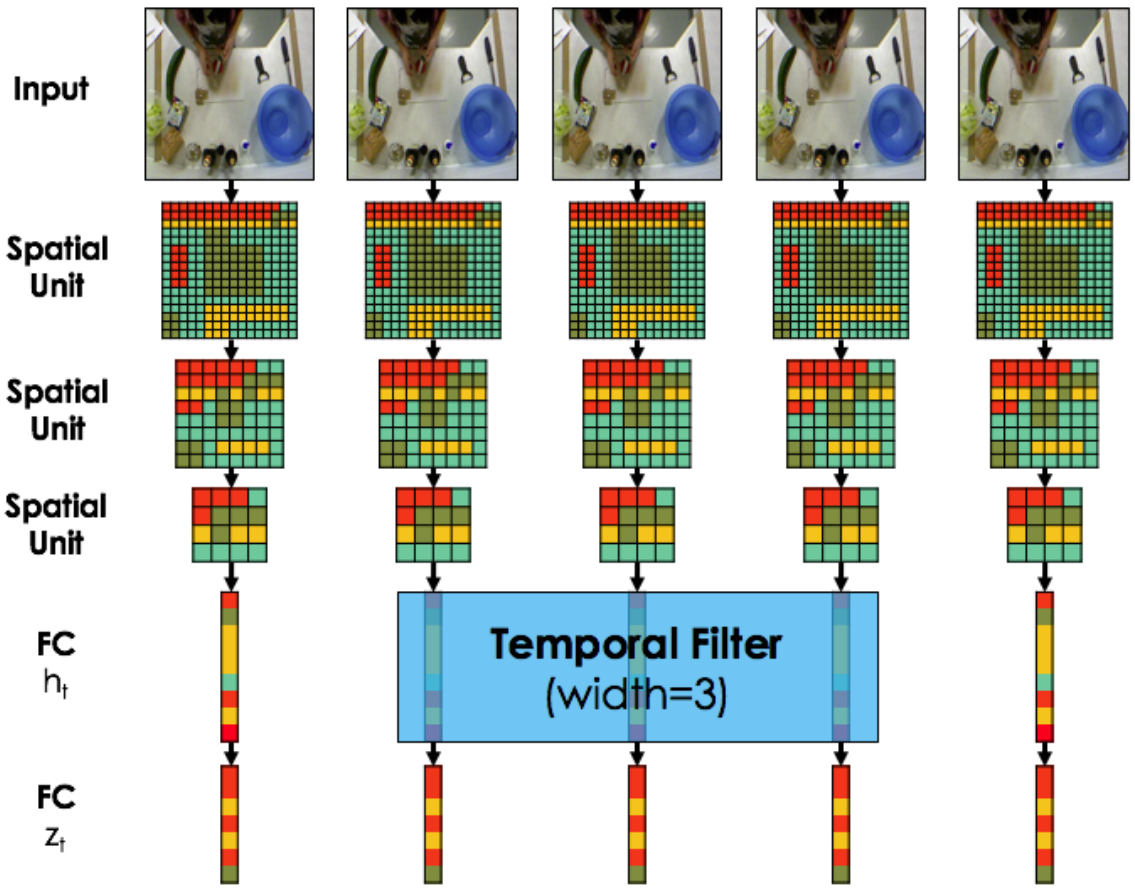


Figure 4.8: The temporal model captures how latent object relationships change over time.

capture properties such as the scene configuration at the beginning or end of an action and different ways users perform the same action. These are similar to the action primitives in the previous chapter, except that we now take a linear combination of shared temporal filters instead of maximizing over a set of class-specific filters. The temporal component is depicted in Figure 4.8.

For a video with duration T, let $H = \{h_t\}_{t=1}^{T} \in \mathbf{R}^{T \times F_{fc}}$ be the matrix of intermediate fully connected features from the spatial CNN, where $h_t \in \mathbf{R}^{F_{fc}}$, and $y_t \in$

$\{1, ..., C\}$ be an action label at time t. We learn F_e temporal filters $W^{(2)} = \{W_1^{(2,i)}\}_{i=1}^{F_e}$ with biases $b^{(2)} = \{b_i^{(2)}\}_{i=1}^{F_e}$. Each filter is of duration d such that the i-th filter is $W^{(2,i)} \in \mathbf{R}^{d \times F_{fc}}$ and corresponding bias is the scalar $b_i^{(2)} \in \mathbf{R}$ for $i \in \{1, \ldots, F_e\}$. The activation matrix $A \in \mathbf{R}^{T \times F_e}$ is given by the convolution of the spatial features H and the temporal filters using a ReLU non-linearity:

$$A = \text{ReLU}(W^{(2)} * H + b^{(2)}). \tag{4.9}$$

Probability vector $\hat{y}_t \in [0, 1]^C$ is a function of weight vectors $W^{(3)} \in \mathbf{R}^{C \times F_e}$ and biases $b^{(3)} \in \mathbf{R}^C$ with the softmax function:

$$\hat{y}_t = \text{softmax}(W^{(3)} A_t + b^{(3)}). \tag{4.10}$$

We choose filter lengths spanning on the order of 10 seconds of video. This is much larger than in related work (e.g. [48, 65]). Qualitatively, we found these filters capture states, transitions between states, and attributes like action duration. In the next chapter we introduce a deep temporal model using similar temporal convolutional filters.

Ideally, the spatial and temporal components of our CNN should be trained jointly, but this requires an exorbitant amount of GPU memory. The activations alone for each image of the spatial network takes approximately 2 Mb. A breakdown of the amount of memory and number of parameters per layer from the spatial and temporal

components are shown in Table 4.1. One minute of footage at full frame rate (30 Hz) is over 3.3 Gb. The maximum length video for 50 Salads, which is 9 minutes and 11 seconds, would be 17 Gb just to store the spatial activations for each layer. Ultimately, we train the spatial network and then the temporal network. We follow the same learning procedure as with the spatial CNN component where we use the categorical cross entropy loss using ground truth labels y. We optimize this function using SGD with ADAM step updates. In our experiments we use $F_e = 64$ temporal filters unless otherwise stated. Below I describe alternative approaches of training that could be done, albeit which may take much more time computationally and would be much less memory efficient.

One way to train the model, which would be more memory efficient than the naive approach, would be to break each video into overlapping sections that are each the length of the temporal filters (e.g., 10 seconds). Using this approach, storing the spatial and temporal activations for this subset of a video would take 306.1 Mb of space, as shown in Table 4.1. If we used a batch size of 32, which is relatively small compared to typical spatial networks, this would require 9.8 Gb of GPU memory. A batch size of 64, which is more common, would be 19.6 Gb. For reference, we used a Nvidia GTX Titan X for most experiments, which has 12 Gb of GPU memory. Note that some of this memory also needs to be allocated for other applications, such as running the graphics for the monitor.

While in theory we could use this approach – especially if we subsampled the

Spatial CNN	Res (per side)	Input channels	Memory (bytes)	# Params
input	108	6	69984	0
conv1	108	64	746496	3456
relu1	108	64	0	0
pool1	36	64	82944	0
conv2	36	96	124416	55296
relu2	36	96	0	0
pool2	12	96	13824	0
conv3	12	128	18432	110592
relu3	12	128	0	0
pool3	4	128	2048	0
FC	1	128	128	16384
output	1	10	10	1280
Spatial total			1.00 Mb	187008
Temporal CNN				
Input	300 (10 sec/30 Hz)	128	(from above)	0
Conv	38400	64	2457600	2457600
FC	1	10	640	640
Temporal total			2.34 Mb	2458860
ST-CNN total			306.12 Mb	2645248
Batch size	64		19.59 Gb	

Table 4.1: Memory and parameter layout of our spatial CNN model. Res. is the resolution (width & height). The 6 input channels refer to RGB and 3 difference images. Memory refers to storage space for the activations for that layer, which must be stored in order to perform backpropagation. # Params refers the number of parameters for all of the weights in that layer. The temporal CNN assumes data is processed at full frame rate (30 Hz) with temporal filters that are 10 seconds long.

images spatially or temporally – it results in a substantial increase in the amount of data that needs to be read into memory. In our current approach, we store as many raw images into RAM (non-GPU memory) as possible and then during training load these batches on the GPU. In this proposed approach we would have to load a video into RAM, temporally oversample that video to create our batches of length d, and then during training send these batches to the GPU. This results in much more redundancy because we oversample the video to create each batch and creates a large bottleneck on input-output. As a reference point, reading from a hard drive is on the order of a magnitude slower than from RAM. In this approach we would have to read much more often from RAM.

A third way of training the model would be to alternatively optimize the spatial and temporal networks. First, one would train the spatial network for a small number of iterations (e.g., 5 epochs), then train the temporal network for a small number of iterations, and iterate back and forth until convergence. We could use the same pipeline that we currently have to train the spatial network, and modify it slightly to incorporate the activations from the temporal component. We leave this approach to future work.

Temporal Filters versus Action Primitives: The action primitives from Chapter 3 and the temporal filters defined in this chapter are conceptually similar, but have a few notable differences. First, in the LC-SC-CRF we used a set of per-class action primitives whereas now we use a set of filters that are shared amongst all

classes. Shared filters may be more appropriate because certain actions (e.g., cut tomato versus cut cucumber) may contain the same subactions. Computationally it is more efficient to share filters, especially if there is a large number of action classes. Second, the score for a given time step in the LC-SC-CRF was computed by maximizing over the scores of each primitive (of a given class) whereas scores using the new architecture are computed by taking linear combinations of all filters. Thus, now we blend together different filters. For example, a given score may be a linear function of consecutive subactions such as picking up the knife and cutting the tomato. The last major difference is that previously we learned the primitives using the Latent Structural SVM loss function whereas now we are using the cross entropy loss. The latter is more common within the CNN literature; it is unclear whether there is any performance benefit from using either function.

4.4 Baselines

We compare against two spatial baselines on both datasets using Improved Dense Trajectories and a pre-trained VGG network.

Hand-crafted Features: The Improved Dense Trajectory (IDT) baseline is comparable to Rohrbach *et al.* [49] on the MPII dataset. IDT features are computed as follows with the publicly available code from the authors[5]. Using dense optical flow, extract all tracklets from a video, where each tracklet consists of a continuous tra-

[5] `https://lear.inrialpes.fr/people/wang/improved_trajectories`

jectory over 15 consecutive frames (0.5 seconds). Some of these tracklets are filtered out in order to account for camera motion using a homography-based approach with SURF features. Spatial and spatiotemporal texture features such as Histogram of Oriented Gradients (HOG), Histogram of Optical Flow (HOF), and Motion Boundary Histograms (MBH) are extracted at the beginning, middle, and end of each tracklet along with its corresponding x and y position within each frame. These features are used in a Bag of Words (BoW) model using KMeans or Fisher Vector encodings. The output for each frame is either a histogram computed from the learned KMeans model or a Fisher Vector. Typically the BoW dictionary is very large (e.g. $k = 1000$) so the dimensionality of the feature vector is also large. We extract IDT, create a KMeans dictionary ($k = 2000$), and aggregate the dictionary elements into a locally normalized histogram with a sliding window of 30 frames. We only use one feature type, HOG, because it outperformed all other feature types or their combination in preliminary experiments. This may be due to the large dimensionality of IDT and relatively low number of samples from our training sets. Note that it took 18 hours to compute IDT features on 50 Salads compared to on the order of 10 minutes for our spatial CNN features using a Nvidia GTX Titan X graphics card. It is trivial to decrease the CNN time by sub-sampling, whereas due to the optical flow computations it becomes more difficult for IDT.

Deep Spatial Features: We also compare against a CNN baseline using the VGG network [95] which has been trained on ImageNet. We use the activations from FC6,

the first of VGG's three fully connected layers, as the features at each frame. We performed preliminary experiments using other layers of activations and found FC6 outperformed others. We also experimented fine-tuning the last few layers of the pre-trained VGG network. We do not show a the fine-tuned results; performance was notably lower than our spatial CNN.

For our spatial-only results, we classify the action using the IDT or VGG features at each time step with a linear Support Vector Machine.

Temporal Classifier: The temporal information captured by the ST-CNN is relatively local; it only captures 5-10 seconds worth of information. We compare performance using our CNNs as input into our previous models. In particular, we compare performance using just the Spatial CNN, just the ST-CNN, ST-CNN with K-Segment inference, and the Spatial CNN with the LC-SC-CRF model. We denote the models using segmental inference as S-CNN+Seg or ST-CNN+Seg.

When using the Spatial CNN with the LC-SC-CRF model, the input is the set of log probabilities of the output of the Spatial CNN, $\log(\hat{z}_t)$, for each time step. All additional parameters, such as the pairwise transitions, are learned using the SSVM. We use $H = 3$ latent states in the LC-SC-CRF. As in Chapter 3, for JIGSAWS we set $d = 100$ and for 50 Salads we set $d = 200$.

Spatial & Handcrafted Models	**Edit**	**Accuracy**
VGG	7.6	38.3
IDT	16.8	54.3
S-CNN	**25.5**	**68.0**
Spatiotemporal Models	**Edit**	**Accuracy**
S-CNN + Seg	47.8	70.3
ST-CNN	52.8	71.3
ST-CNN + Seg	**53.7**	**71.6**

Table 4.1: 50 Salads results using our video-based models on the higher-level action granularity. As expected, these results are lower than the sensor-based results in Chapter 3.

4.5 Experiments

4.5.1 Video-only Results

We evaluated several variations of our spatial and spatiotemporal models on 50 Salads and JIGSAWS using segmental edit distance and frame-wise accuracy. We also assessed action classification performance – the case where we know the temporal boundaries – using segment-wise accuracy.

Tables 4.1 and 4.2 show segmentation results using IDT, VGG, and our models. S-CNN, ST-CNN, and ST-CNN + Seg refer to the spatial, spatiotemporal, and segmental components of our model. These 50 Salads results are on the "higher" granularity. Our full model has 17% better accuracy on 50 Salads and 23% better accuracy on JIGSAWS compared to the IDT baseline. Figures 4.9 and 4.10 shows example predictions using each component of our model.

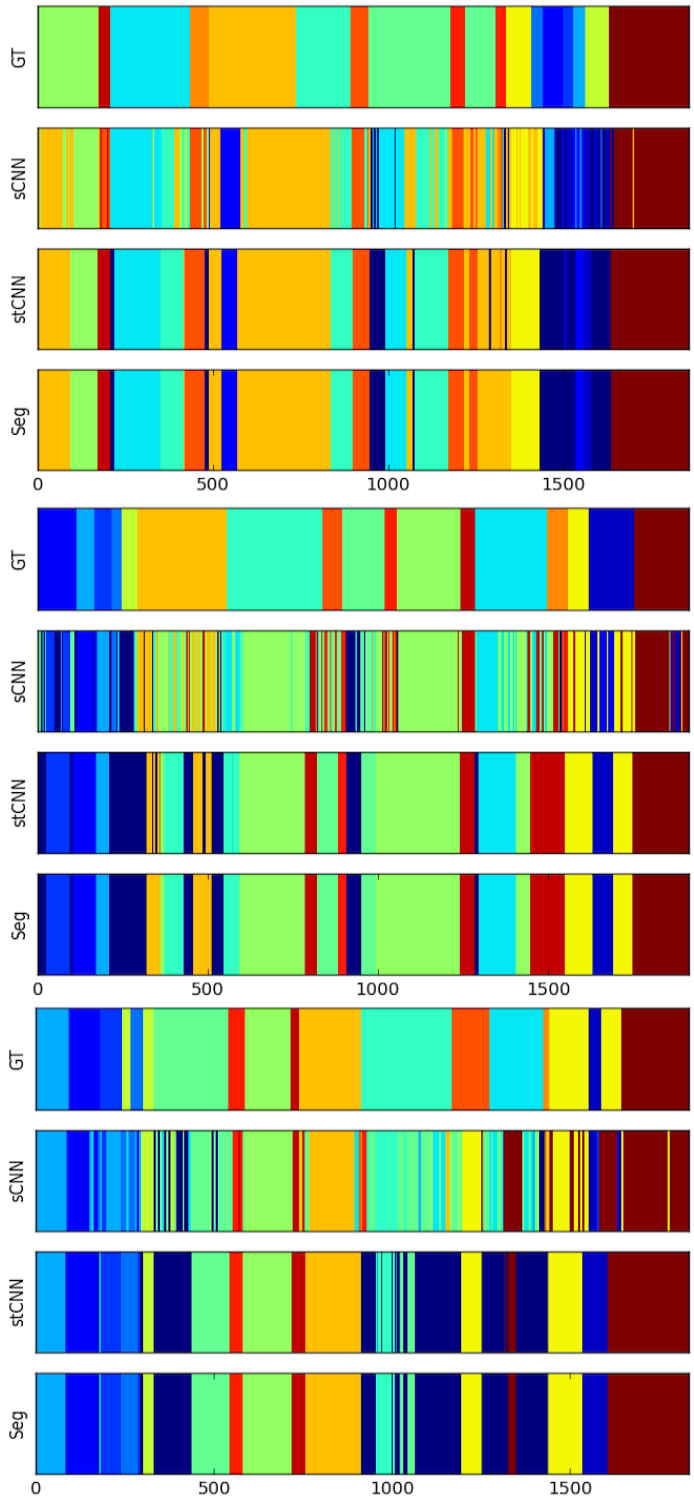


Figure 4.9: The top plot for each video depicts the ground truth action at each time step for a video in 50 Salads. Subsequent rows show predictions using the Spatial CNN, Spatiotemporal CNN, and Spatiotemporal CNN with segmental inference. Each color corresponds to a different class label.

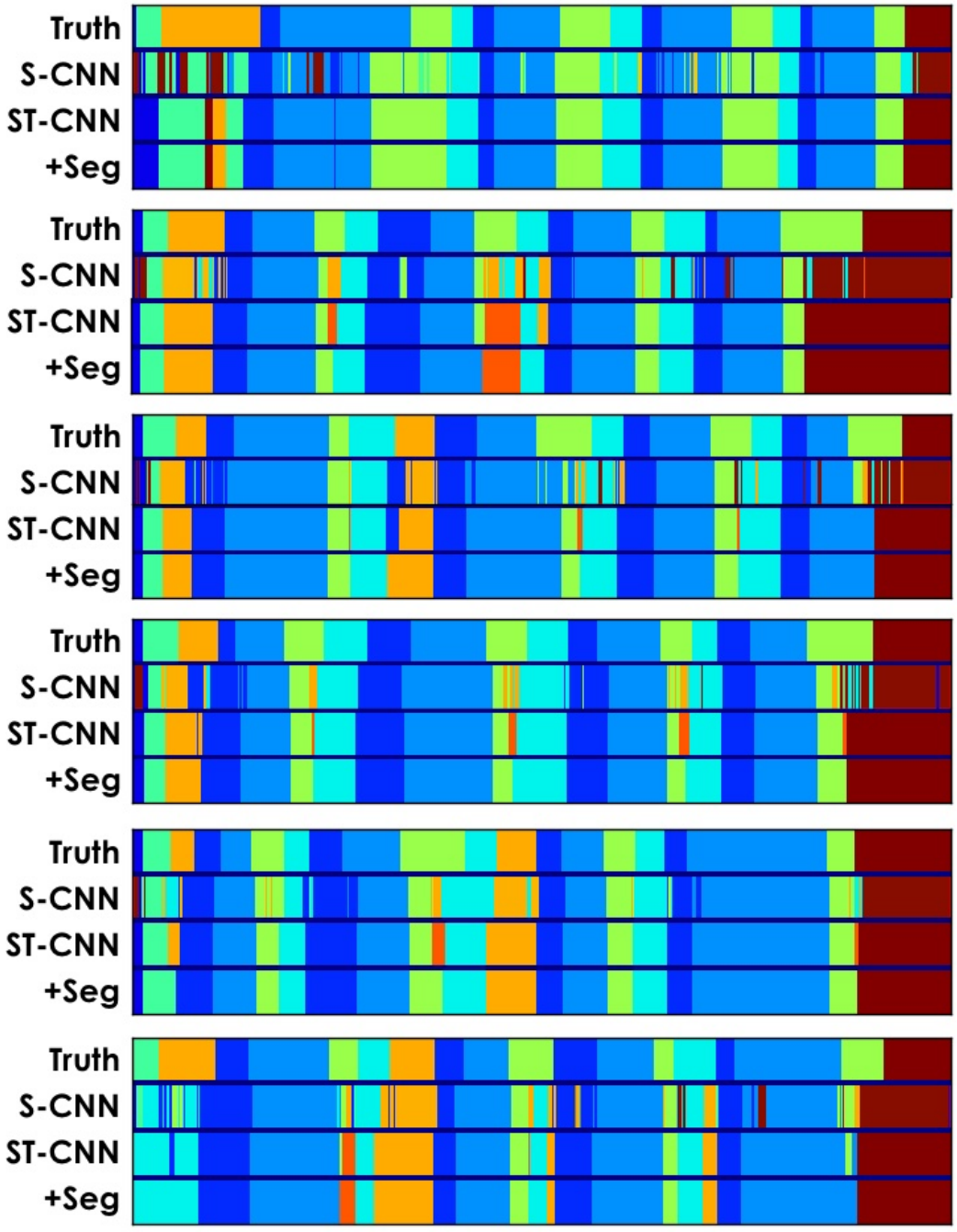


Figure 4.10: The top plot for each video depicts the ground truth action at each time step for a video in JIGSAWS. Subsequent rows show predictions using the Spatial CNN, Spatiotemporal CNN, and Spatiotemporal CNN with segmental inference. Each color corresponds to a different class label.

Spatial & Handcrafted Models	**Edit**	**Accuracy**
VGG	24.3	45.9
IDT	8.5	53.9
S-CNN	**37.7**	**74.0**
Spatiotemporal Models	**Edit**	**Accuracy**
[33] STIPS+CRF	-	71.8
S-CNN + Seg	53.3	74.3
ST-CNN	62.0	**77.3**
ST-CNN + Seg	64.4	**77.3**
LC-SC-CRF	**72.7**	72.6

Table 4.2: JIGSAWS results using our video-based models. As expected, these results are lower than the sensor-based results in Chapter 3.

In the first experiments, we highlight how effective the baseline models are at capturing actions using either one frame or the bag of words features computed in a local window around that frame. While our results are still insufficient for many practical applications the accuracy of our spatial model is at least 14% better than IDT and 28% better than VGG on both datasets. One large reason for this is in the large number of parameters each model contains, which results in extreme overfitting. For VGG, the final classifier has approximately 138 million parameters and for IDT it has on the order of 1 million. Our model only has on the order of 300 thousand parameters. Furthermore, in our case the dimensionality of IDT right before classification stage is 96,000 features, which is much larger than the 128 features in our spatial CNN. Aside from overfitting, IDT suffers from the fact that its bag of words representation does not encode spatial relationships between features. In both of our datasets spatial information is very important. While VGG does encode spatial infor-

mation, it was pre-trained using data augmentation techniques such as image flipping which introduce unwanted spatial invariance. We tried training VGG from scratch but found that it grossly over fit due to the lack of training data.

Note that the edit score is very low for all spatial models. This is not surprising because each model only uses local temporal information which results in many oscillations in predictions, as shown in Figures 4.9 and 4.10.

The spatiotemporal model (ST-CNN) outperforms the spatial model (S-CNN) on both datasets. Aside from modeling temporal evolution, ST-CNNs also smooth out the predictions. The effect on edit score is substantial and likely due to the large temporal filters. By visualizing these features we see they tend to capture different phases of an action like the start or finish.

Segmental Model: Adding segmental inference provides a notable improvement on JIGSAWS results but only a modest improvement on 50 Salads. By visualizing the results we see that the segmental model can be helpful and harmful. For example, when the predictions oscillate (like in Figure 4.10) the segmental model provides a large improvement. However, sometimes the model smooths over actions, like the background class, when they are short in duration. We find the model we present in Chapter 5 has fewer of these errors.

Temporal Filters versus Action Primitives: Table 4.3 compares the temporal component from this chapter with the temporal action primitives from the previous chapter. Note that the latter does not use a temporal prior or pairwise skip chain term.

50 Salads	**Sensor Accuracy**	**Video Accuracy**
Action Primitives	71.95±6.39	70.66±10.76
Temporal CNN	78.67±5.99	70.44±11.12
JIGSAWS	**Sensor Accuracy**	**Video Accuracy**
Action Primitives	73.02±9.40	76.60±8.74
Temporal CNN	76.69±9.66	77.08±8.91

Table 4.3: Comparison of the Action Primitives from Chapter 3 and the temporal CNN filters described here. The sensor results are obtained using the kinematics/accelerometer data from Chapter 3 and the video results are obtained use the spatial CNN activations from this chapter. The action primitives model was learned using the LC-SC-CRF without the temporal skip chains or temporal prior.

Labels	**Classes**	**Edit**	**Accuracy**	**Accuracy** (pre-segmented)
Low	52	29.30	44.13	39.67
Mid	18	48.94	58.06	63.49
Higher	10	66.44	72.71	86.63
Highest	4	83.20	92.43	95.14

Table 4.4: 50 Salads results using all four action granularities with the ST-CNN. The last column (pre-segmented) implies that we took the known start and stop time and took the segment-wise accuracy given the maximum scoring class in that segment.

In both cases we used approximately 30 filters per model (3 per class). Recall that the action primitives were defined per-class and our new temporal filters are shared between classes. The hyper-parameters for both models (e.g., filter duration) were the same; the filter duration was a function of the mean action duration computed over the training data. We performed two experiments: one using the spatial CNN features as input and the other using the sensor data used in Chapter 3. Accuracy using both methods was similar on the video data but there was a large improvement when using the sensor data. Specifically, the temporal CNN filters achieved 7% better performance on 50 Salads and 3% better performance on JIGSAWS.

Action Granularity: Table 4.4 shows performance on all four action granularities from 50 Salads using our full model. The duration of each temporal filter is defined as the mean duration of all action segments. Columns 3 and 4 show scores for segmental and frame-wise metrics on the action segmentation task and the last shows action classification accuracies which assume temporal segmentation is known. While performance decreases when assessing finer granularities, this decrease is not very severe. For example, the difference in accuracy between the mid level (18 classes) versus the low level (52 classes) is only 18%. Some errors at the finer levels are likely due to temporal shifts in the predictions. Given the high accuracy at the courser levels, future work should look at modeling finer granularities by modeling actions hierarchically.

For the classification results, the (known) start and end times are fed into the segmental model to predict each class. Our classification accuracy on JIGSAWS is 90.47%. This is notably higher than the state of the art [142] which achieves 81.17% using a video-based linear dynamical system model and also better than their hybrid approach using video and kinematics which achieves 86.56%. These surgical actions can be recognized well using position and velocity information [1], thus our ability to capture object relationships may be less important on this dataset.

4.5.2 Sensor Substitution Results

In this section we evaluate the performance of the regressed sensor values and the action segmentation using these values on 50 Salads and JIGSAWS. Note, in our

original sensor substitution publication [143] we used a spatial CNN that was similar in spirit but had a slightly different topology than what we described in this chapter. The biggest difference is the input in that paper only used a color image, which had a negative affect on the 50 Salads dataset. The JIGSAWS results are comparable with both models, and thus for convenience these results are taken directly from that paper.

4.5.2.1 Kinematics Results

We performed two types of experiments for JIGSAWS. First, we trained and tested our models using the appropriate task (e.g. Suturing, Needle Passing, or Knot Tying). Second, we trained on each one of the tasks and test on the other two tasks to assess our ability to generalize across tasks. For example, we trained on suturing and tested on knot tying.

Kinematics Prediction: Table 4.5 shows the root mean squared error for all combinations of training and testing sets. Figure 4.11 shows the predicted positions and ground truth plots of the x, y and z coordinates of each gripper for a Suturing trial. The prediction – while noisier – follows the ground truth closely except when it spikes, for example when the tool goes out of view of the camera.

The velocities, shown in Figure 4.12, tend to follow the same patterns as the true sensor values, but predicted values tend to underestimate the signal. Perhaps if we used another type of motion image, such as optical flow, we would improve velocity

train	test	left x	left y	left z	right x	right y	right z
SU	**SU**	**0.43**	**0.33**	**0.54**	**0.85**	**0.88**	**0.77**
KT	SU	0.62	0.78	0.86	1.37	1.12	1.22
NP	SU	1.00	1.28	1.30	2.64	2.13	1.76
SU	KT	0.74	0.74	1.34	1.13	1.00	1.20
KT	**KT**	**0.44**	**0.39**	**0.59**	**0.52**	**0.34**	**0.67**
NP	KT	0.87	1.25	1.22	1.21	1.21	0.94
SU	NP	1.98	0.95	1.22	2.19	0.87	1.22
KT	NP	2.06	1.18	1.28	2.04	0.77	2.12
NP	**NP**	**0.80**	**0.50**	**0.95**	**0.90**	**0.65**	**0.92**

Table 4.5: Root mean squared error of the predicted kinematics in centimeters. Performance of each task is averaged over cross validation splits and repeated for each of the tasks.

performance.

Figure 4.13 shows the gripper state prediction for each tool and Figure 4.14 shows the corresponding ROC curves for *is_open* and *is_closed*. Visually, the gripper represents a small part of each image, so it is not surprising that the results are much noisier than the position or velocities. Despite this noise the predictions follow the general pattern as the ground truth. The average AUC score for both tools was 0.8026.

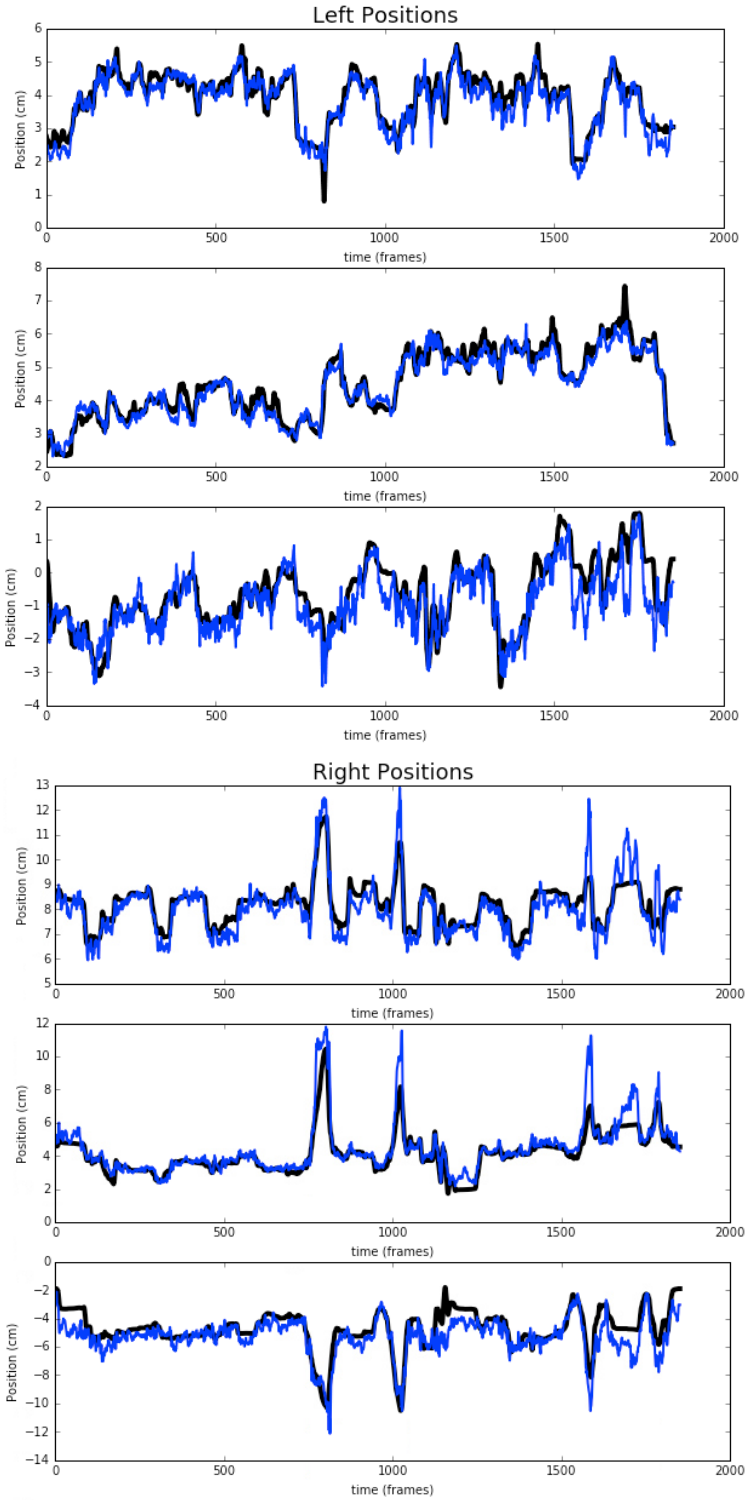


Figure 4.11: The six plots show ground truth (black) and prediction (blue) of x, y and z position of the right and left gripper (in centimeters). The network was trained and tested on the Suturing task.

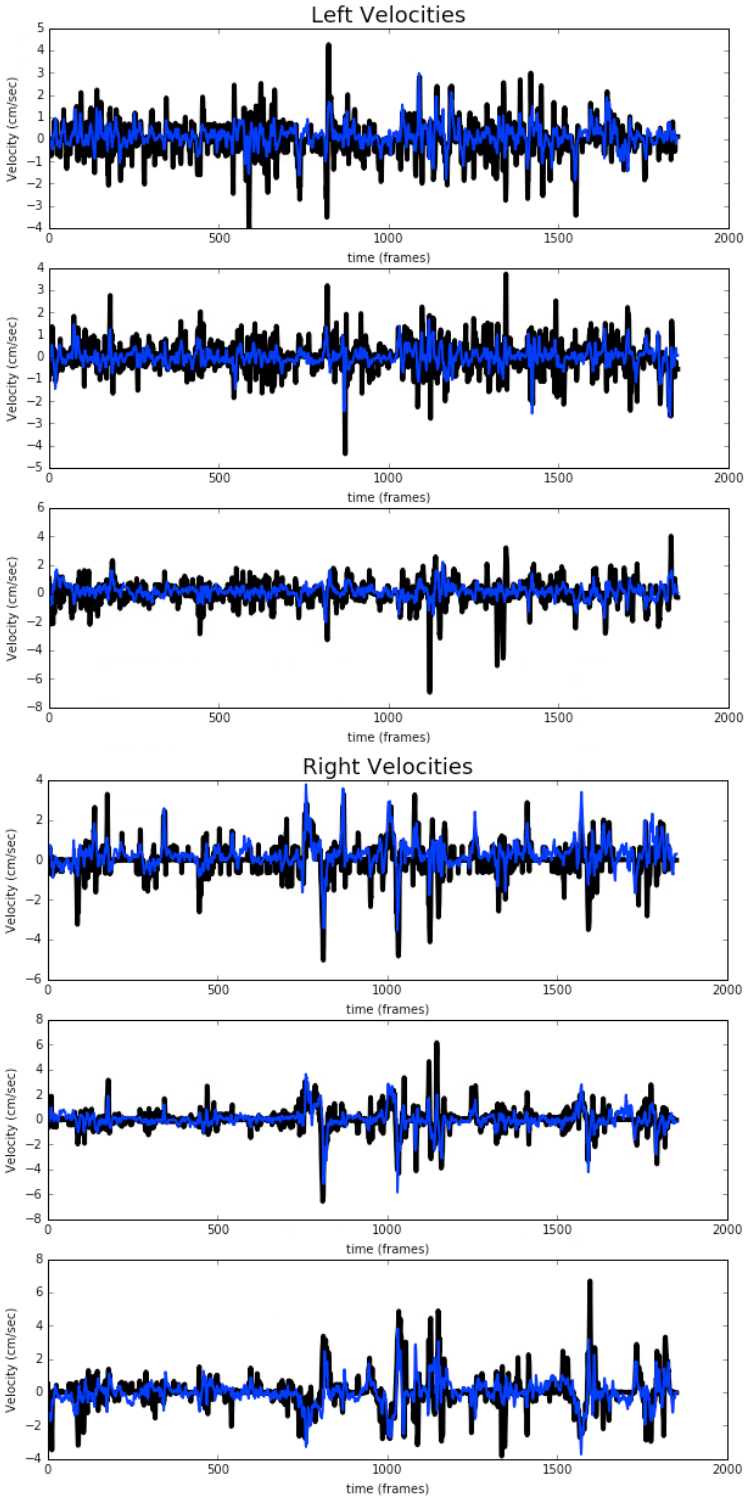


Figure 4.12: The six plots show ground truth (black) and prediction (blue) of x, y and z velocity of the right and left gripper (in centimeters/sec). The network was trained and tested on the Suturing task.

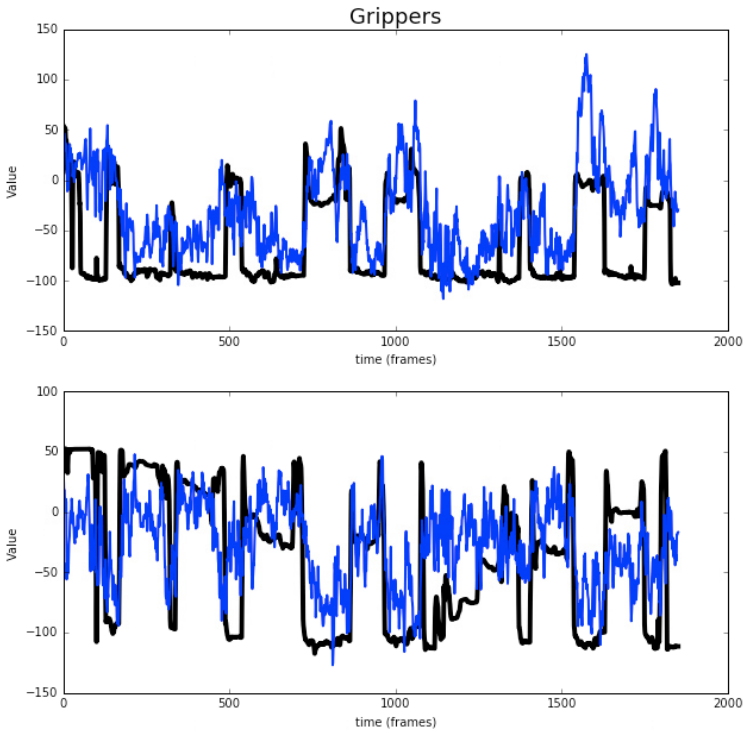


Figure 4.13: These two plots show ground truth (black) and prediction (blue) of the gripper angle for each tool. The network was trained and tested on the Suturing task.

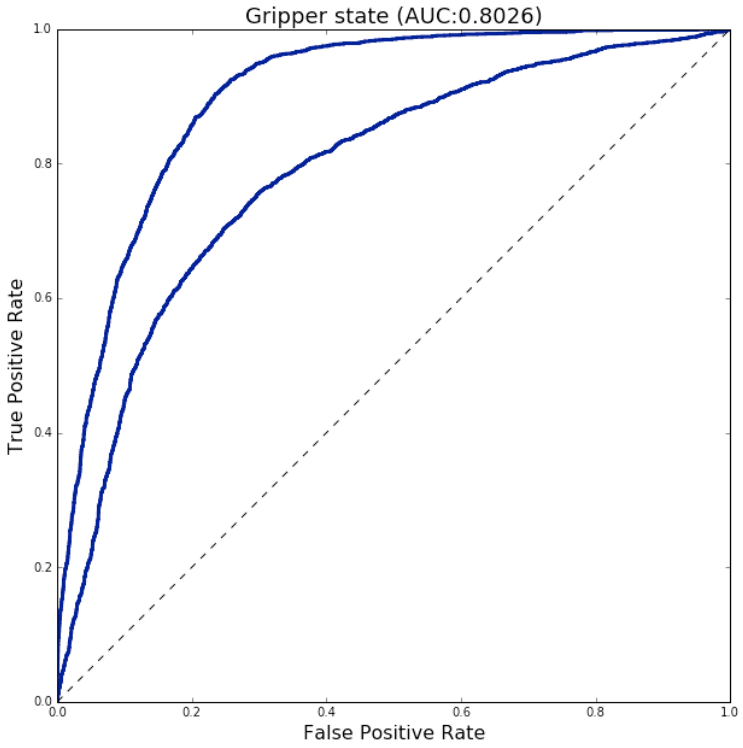


Figure 4.14: An ROC curve for each of the two grippers being *open* or *closed*. A gripper is considered open if the true value is above a set threshold τ.

When training the CNN on one task (e.g., suturing) and evaluating on a second (e.g., knot tying) the predicted positions followed the correct general pattern, but it appeared to be transformed and scaled in 3D space. Figure 4.15 shows example position predictions. The scaling is likely due to the fact that the camera is situated at a different position with respect to the tools in each task. Given this transformation between tasks, we did not expect to predict the true positions, but did expect to predict the general spatial patterns. While this scaling and rigid transform may be problematic for some applications, it may not be problematic for action segmentation; it may be sufficient to rely on the relative positions within a task.

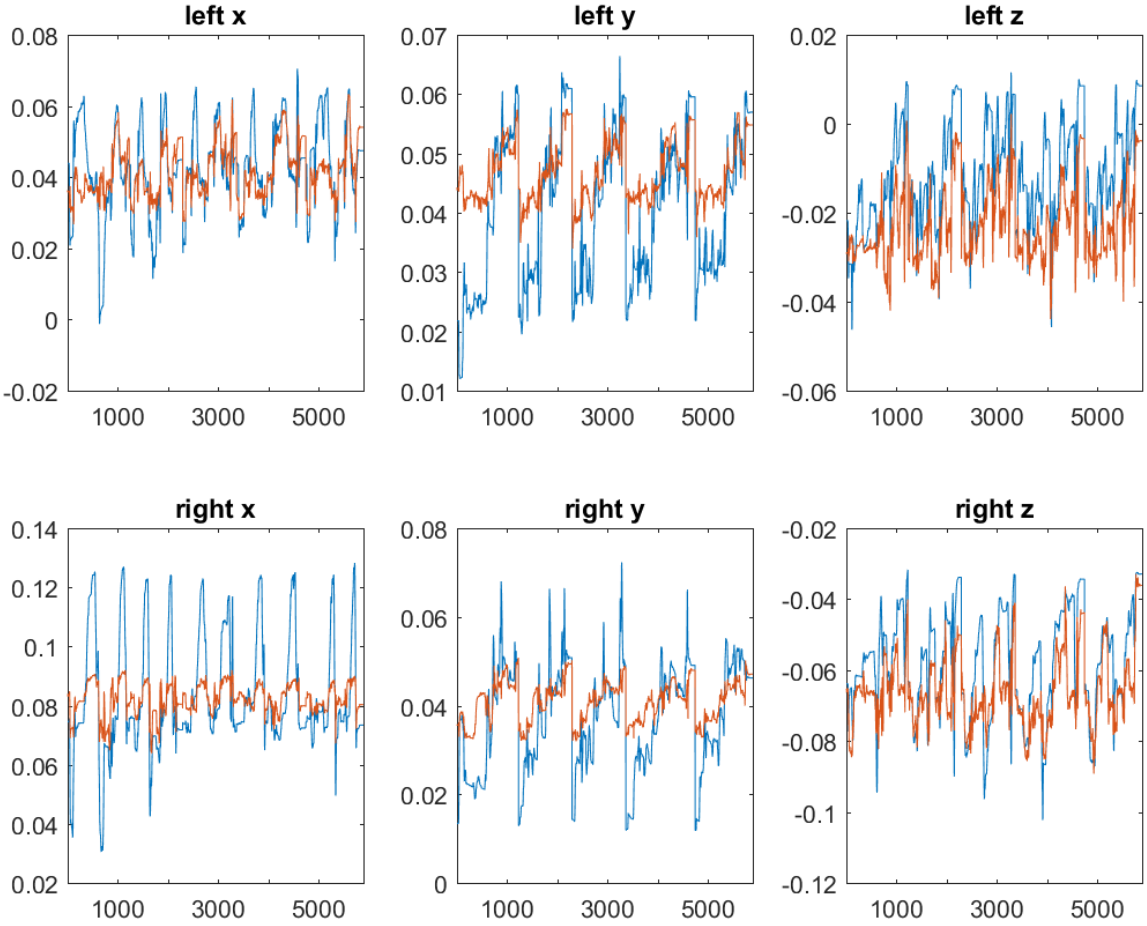


Figure 4.15: Predictions (orange) of a network trained on suturing tested on a sequence of the knot tying task. Ground truth (in meters) is shown in blue. We see that there is a shift in the global coordinate system between the two tasks but still the relative motion is predicted accurately.

Action Segmentation: We used the predicted gripper positions as input into our LC-SC-CRF from Chapter 3 to assess our action segmentation performance. Table 4.6 shows results of our model trained and tested on all nine combination of the three JIGSAWS tasks as well as those using the ground truth positions as input. We also show the results of Tao *et al.* [33] and our suturing results from the ST-CNN with segmental inference.

Test Task	GT	SU	KT	NP	ST-CNN+Seg	[33]
Suturing	76.14	**76.64**	65.84	61.20	77.30	71.75
Knot Tying	74.27	70.37	**76.35**	67.82	-	66.94
Needle Passing	62.37	51.34	54.21	**60.46**	-	60.39

Table 4.6: Action recognition results on JIGSAWS. Columns refer to models trained on ground truth kinematics (GT) or predicted from a CNN trained on each task (SU, KT, or NP). The ST-CNN+Seg results are from Table 4.2.

Our model outperformed Tao *et al.* [33] on all three tasks. On Suturing and Knot Tying ours performed notably better and for Needle Passing ours achieved comparable results. Needle Passing is visually very different from the other two tasks which makes it more challenging to learn the appropriate information. This also explains why training on Needle Passing gives slightly higher error in Table 4.5 than when training on the other tasks. For suturing and knot tying our model achieved slightly higher accuracy using the predicted kinematics than when using the ground truth signals from the robot. It is possible that the CNN is learning a better basis for the position data, however, the difference in accuracy is not substantial enough to validate this hypothesis.

4.5.2.2 Accelerometer Results

The accelerometer predictions originally reported in [143] were much lower than the ones we report here. The key difference is that here we use both the color and difference images as input whereas that paper only used color images.

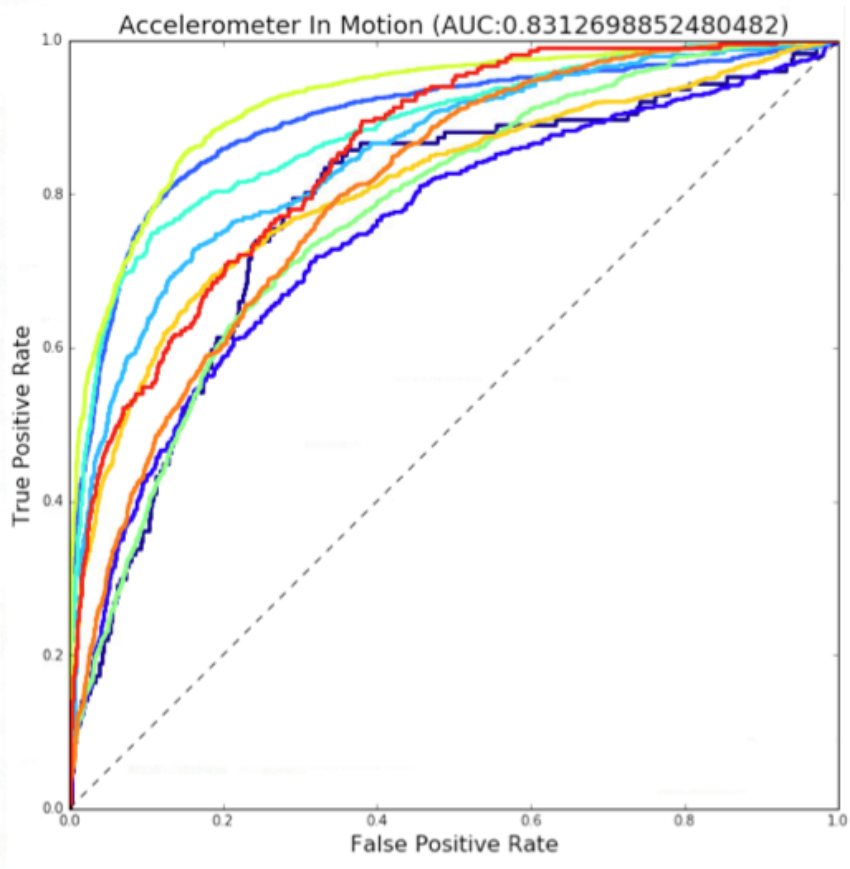


Figure 4.16: ROC curve for each tool in the 50 Salads dataset.

Tool motion prediction: We predict real-valued accelerometer values, however, the results are very noisy. To simplify the problem we assess performance based on whether the tool is predicted as `in_motion` or not `in_motion`. Recall that the label `in_motion` implies that there is non-zero acceleration for a given sensor. While this is not as nuanced as evaluating on the real-valued signals, it is still informative for our task, as we show in the action segmentation results.

Figure 4.16 shows an ROC curve for each of the 10 tools: `pepper dispenser`, `bowl`, `oil bottle`, `large spoon`, `dressing glass`, `knife`, `peeler`, `small spoon`,

`plate` and `chopping board`. Figure 4.17 shows the AUC for each of the tools independently. On average the AUC across tools was 0.8313.

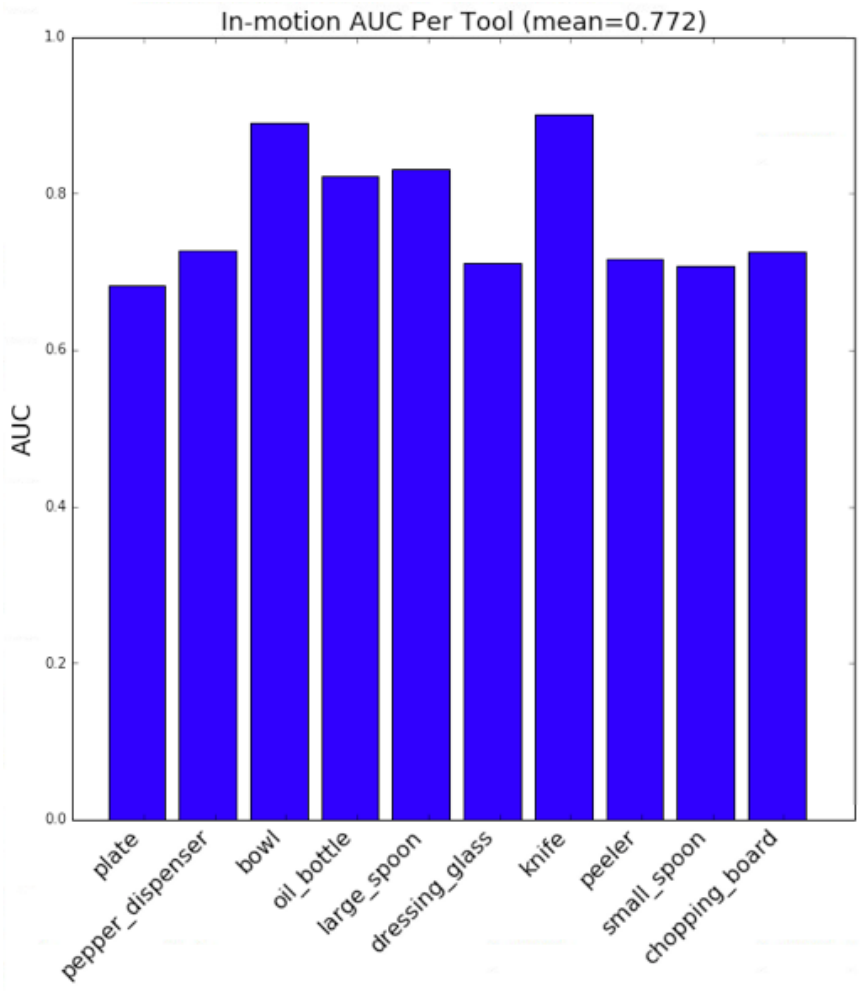


Figure 4.17: AUC score for each tool in the 50 Salads dataset.

Figure 4.18 shows an example set of accelerometer predictions. Each set row corresponds to the magnitude of true or predicted accelerometer for each tool. There appears to be a large amount of noise, however, our method tends to pick up on the dominant pattern for each tool.

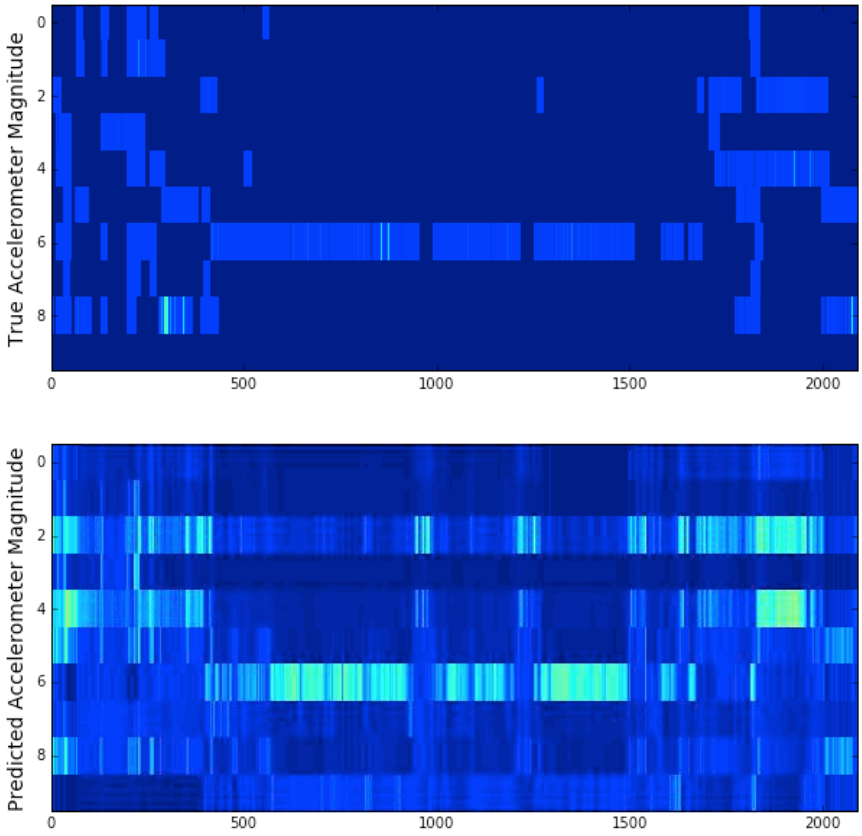


Figure 4.18: True and virtual accelerometer values for a video in the test set of 50 Salads. Dark blue indicates no motion and lighter blue and green indicates positive acceleration. Each row corresponds to a different tool in the order: `pepper dispenser`, `bowl`, `oil bottle`, `large spoon`, `dressing glass`, `knife`, `peeler`, `small spoon`, `plate` and `chopping board`

Action Segmentation: The goal of this thesis is to improve methods for action segmentation, so in the following experiments we show the impact of using a spatial CNN trained using sensor data versus trained solely using action labels. We use the LC-SC-CRF – with filter duration and skip length equal to the mean action segment duration – with four feature types. First, we use the *in motion* versus *not in motion* indicators from the real accelerometer data. Second, we use the virtual

50 Salads Features	**Accuracy**
True Sensors	76.15
Virtual Sensors	70.74
FC from Sensor-trained CNN	70.13
FC from Action-trained CNN	73.50

Table 4.7: Action segmentation results on 50 Salads. True sensors refers to using *in motion* or *not in motion* indicators from the true accelerometer data, virtual sensors refers to the indicators output from the CNN network trained on sensor data, and FC from X refers to using the intermediate fully connected layers from the CNNs trained on sensor data or action labels.

sensors output from the Spatial CNN trained to regress sensor values. Next, we use the intermediate fully connected layer (h_t) from the sensor-trained CNN. Finally, we use the intermediate fully connected layer (h_t) from the spatial CNN trained using only action labels.

Results for all four methods are shown in Table 4.7. While performance using the virtual sensors is not as high as the ground truth, it is close to the performance of the CNN trained using action labels. This implies that we can train a CNN network on a large dataset without action labels, and achieve almost as good accuracy as if we had trained it with these labels.

4.6 Visualization

We visualized our CNNs in three ways corresponding to the video-only CNN, the continuous sensor-trained CNN, and the discrete sensor-trained CNN. We also visualized the IDT approach.

Method 1 (Video-only): We visualized the internal activations of the video-only method by analyzing which filters in each spatial unit have the highest activations. The hypothesis is that the filters with the highest score contribute most to the scores in the next layer. For each unit we show two images: a score image and a filter-index image. The former shows the activation, for each pixel (or region), with the highest value. The latter indicates which convolutional filter gave this score and is colored based on the index of the best scoring filter. Note that each filter corresponds to a different color and that the colored indices in each layer are independent of the colors in other layers. In the score images, the colors range from black (low) to red (neutral) to yellow (high). Thus, objects that are most relevant should be colored yellow.

Figures 4.19 and 4.20 show example activation images. The first corresponds to images taken from a video in 50 Salads. By studying these images and the corresponding videos we can discern the objects and relationships that the CNN captured. An obvious example is as follows. In the first time step, the salad bowl is empty and it receives highest activations from the red and green filters. Over time, as the user adds ingredients to the bowl, other filters (e.g. yellow, orange, teal) have higher activations. This is most clear in the images for Unit 1 and Unit 2.

There are two dominant trends in the JIGSAWS visualizations. First, the activations are highly correlated with the location of the robot end effectors. This is evident in all three time steps and in each of the spatial units. This implies that position of the end effector is a useful property to capture for recognizing actions. Second,

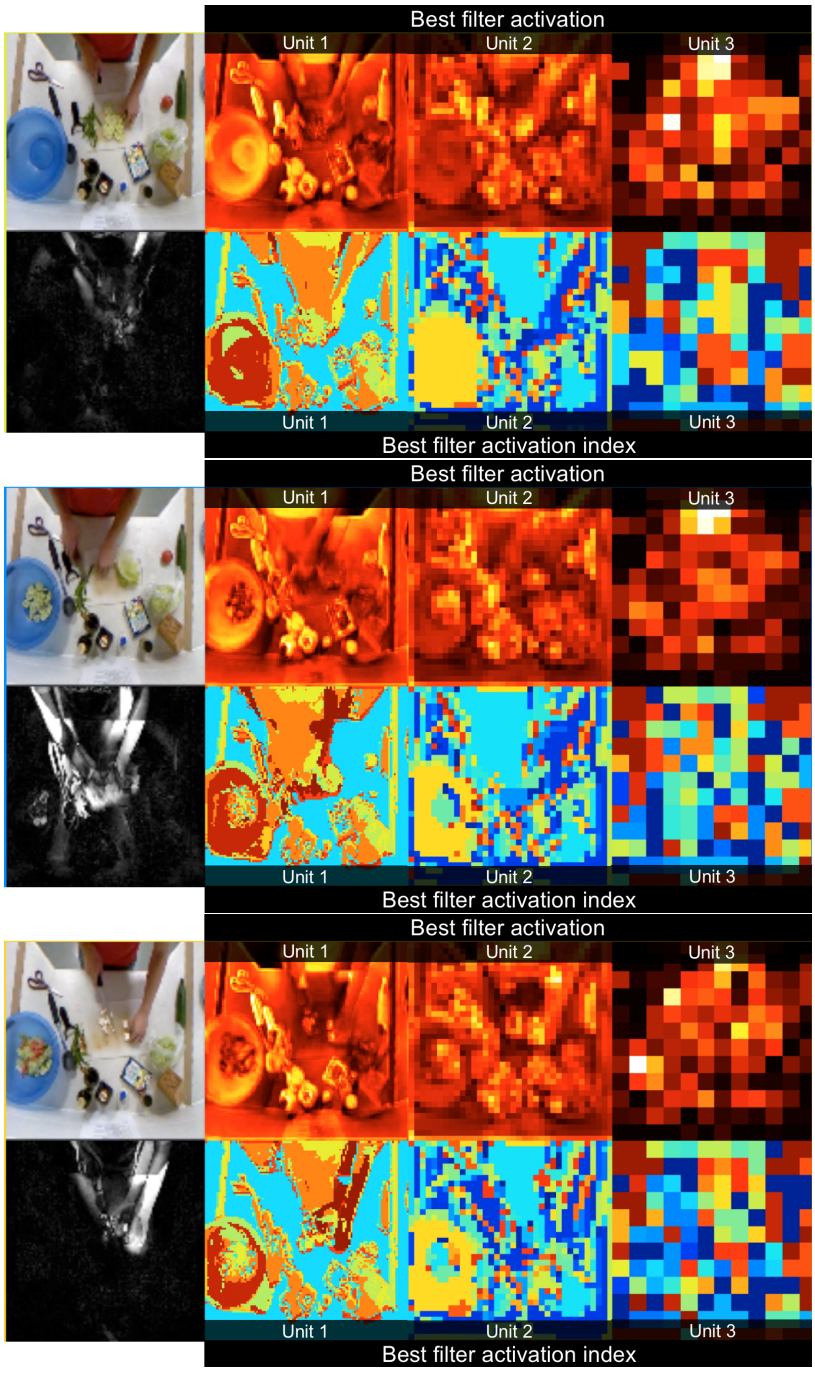


Figure 4.19: Visualization of method 1 for a sequence of images on 50 Salads. The left columns correspond to the input image and corresponding motion image for the given time step. Each subsequent column corresponds to a score- and index- image as described in the text. The colors range from black (low) to red (neutral) to yellow (high)

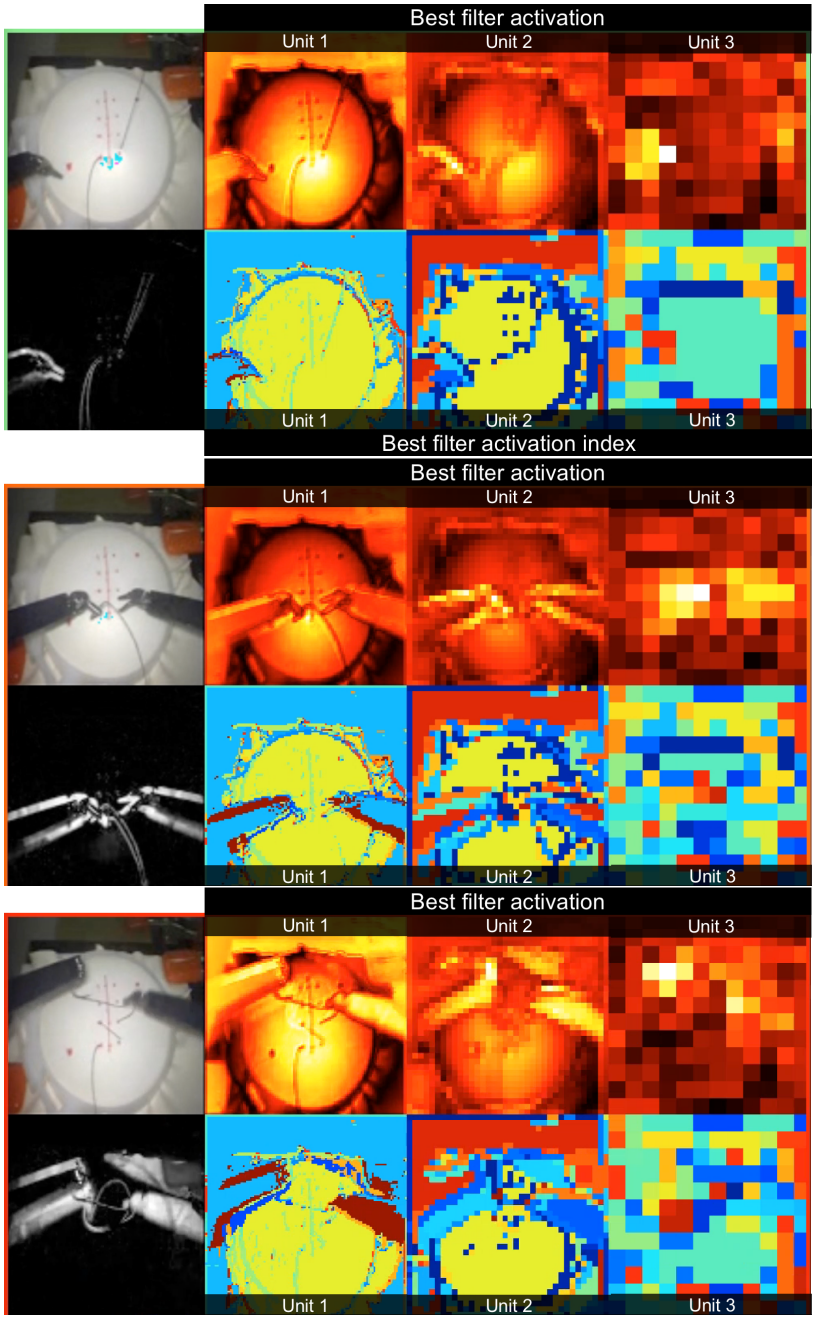


Figure 4.20: Visualization of method 1 for a sequence of images on JIGSAWS. The left columns correspond to the input image and corresponding motion image for the given time step. Each subsequent column corresponds to a score- and index- image as described in the text. The colors range from black (low) to red (neutral) to yellow (high)

whenever a tool tightens the suture thread, the center of the image is highlighted in yellow. This is evident in the first and third image.

Method 2 (Video+Continuous Sensors): For continuous data we adapted the method by Bolei *et al.* [144] to understand which parts of an image are most important for the given prediction. We generate an auxiliary image for each of the N_0^2 regions in the image which we index as I_t^k for $k \in \{1, \ldots, N_0^2\}$. For each region, we occlude that partition of the image and compute a spatial CNN prediction $\hat{z}_t^k$. We compute a difference image $D \in \mathbf{R}^{N_0 \times N_0}$ which measures the change between the un-occluded prediction $\hat{z}_t$ and each occluded prediction $\hat{z}_t^k$:

$$D_k = ||f(I_{t,k}^*) - f(I_t)|| = ||\hat{z}_{tk}^* - \hat{z}_t||. \tag{4.11}$$

This matrix measures the importance that each region has on the final prediction. If the occlusion creates a large change in prediction, it implies that the original information at this location as significant.

Bolei *et al.* occluded each region by adding random noise. Because our camera is static, we found that occluding the image using the median image from the given video was more informative. To be specific, the pixels in occluded region of image I_t^k are replaced by the median value, for each pixel location, from the whole video.

We used this technique to visualize the sensor values corresponding to the left and right grippers in JIGSAWS. Figure 4.21 (top) shows the input image, (middle) shows

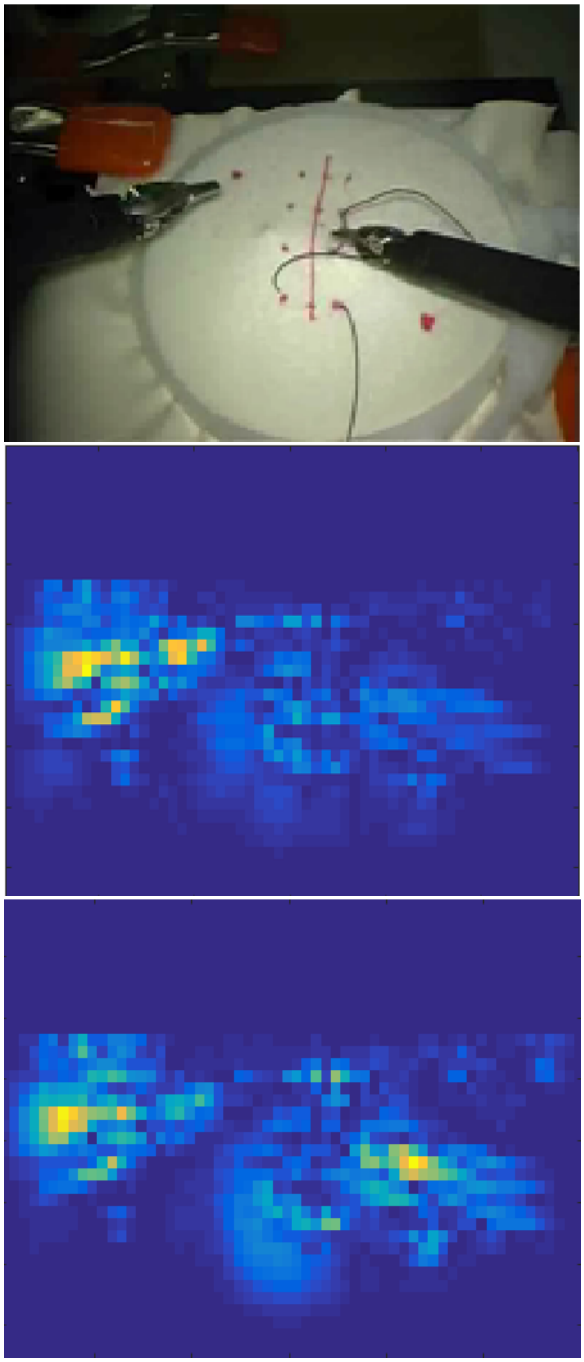


Figure 4.21: Heatmaps for occluding the input image (top) with a sliding gray patch (10×10 pixels) and measuring the change in the prediction of left (middle) and right (bottom) gripper.

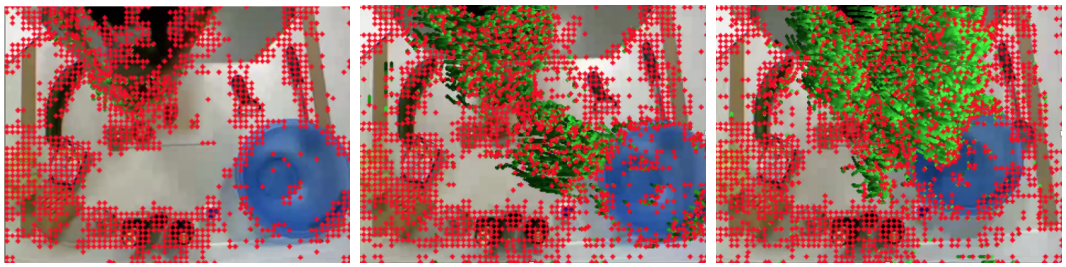


Figure 4.22: Spatiotemporal features are extracted for each of the green tracklets. Note that very few tracklets are detected when there are only very small motions (e.g. left) whereas many tracklets are detected when there are large motions (e.g. right)

the change map corresponding to the left gripper and (bottom) shows the change map corresponding to the right gripper.

Dense Trajectories: Many actions in 50 Salads, like cutting, require capturing small hand motions. We visualized IDT using the public software from Wang *et al.* [9] and found it does not detect many tracklets for these actions. Some examples are shown in Figure 4.22. In contrast, when the user performs actions like placing ingredients in the bowl, IDT generates thousands of tracklets. Even though the IDT histograms are normalized, we find the variation in the number of tracklets is still problematic.

4.7 Conclusion

In this chapter, we extended the ideas from Chapter 3 to the video domain via two primary contributions. First, we factorized video into a spatial component and a temporal component which captures the state of the scene in each image and how this state changes over time. We showed promising quantitative results, which show our model substantially outperforms state-of-the-art action recognition methods, and

showed compelling qualitative results, which highlights that our model is capable of capturing important spatiotemporal patterns.

Our second contribution, sensor-substitution, showed that we can regress position- and acceleration-based sensors with reasonable performance with a variation on our spatial CNN. Using this approach, we showed that we can learn a spatial representation from video without requiring hand-labeled annotations, and furthermore, that we can predict virtual sensors with sufficient performance for some applications outside of action segmentation.

While the performance of our video-based models has not yet reached that of the sensor-based alternatives, we have have make substantial progress relative to our baselines. In the next chapter we will introduce our final model which, in some cases, closes this gap.

Chapter 5

Unified Models for Action Segmentation and Detection

The theme of this chapter is unification. Thus far our models have been comprised of two parts: low-level features (e.g., action primitives) and high-level temporal classifiers (e.g., CRFs). In this chapter we introduce a single hierarchical model that is capable of capturing both components. In addition, we explore the connections between action segmentation and action detection, two areas within computer vision, and show that they are really the same problem. We highlight limitations in the metrics commonly used for each task and introduce a new metric that overcomes these limitations.

5.1.1 Current Limitations

Before continuing, it is worth addressing some of the fundamental limitations with our previous models. Despite achieving large performance gains compared to prior work, certain issues preclude us from using them in some of the applications we care about, including human-robot interaction and some use cases in surgical workflow analysis.

First, our models must be run "offline," meaning we can only predict actions after we have seen the whole video. Many robotics applications require "online" evaluation such that the action is predicted as it happens. For example, in a manufacturing application a robot should be able to interact with the user in real-time.

Second, our models employ a Markov assumption. This is often appropriate for segmentation datasets, like JIGSAWS, where each action is followed by another. However, in detection datasets like MERL Shopping almost all actions have a background segment between them. Background segments may occur when the user pauses to contemplate their next action. This has a large impact on the distribution of pairwise transitions in a chain CRF; with high probability all actions lead to the background class and the background class leads to all other actions. This may be why detection papers tend not to use Markov models.

Third, our models cannot capture some long-range temporal patterns. For the LC-SC-CRF, we assumed that each time step (or segment), conditioned on the pre-

vious action label, was independent of the data at all previous time steps. Action primitives do capture the past d frames, however, the duration was limited to around 10 seconds. To illustrate this issue, consider the 50 Salads dataset. When a user cuts a tomato, the tomato is often occluded by their hands and the knife. If we detected the tomato texture near the hand texture several seconds earlier, and the user is occluding something with the knife, we should be able to infer that they are cutting the tomato. These complex patterns are not captured by our previous models.

Finally, we found that temporal convolutional filters from Chapter 4 significantly outperformed the action primitives from Chapter 3 using sensor data, but we did not propose a good way of incorporating them into the LC-SC-CRF.

In this chapter we introduce a model that addresses all of these issues.

5.1.2 Contributions

We introduce a class of time-series models, which we call Temporal Convolutional Networks (TCNs), that capture long-range patterns using a hierarchy of temporal convolutional filters. We present two types of TCNs: First, our Encoder-Decoder TCN (ED-TCN) only uses a hierarchy of temporal convolutions, pooling, and upsampling but can efficiently capture long-range temporal patterns. The ED-TCN has a relatively small number of layers (e.g., 3 in the encoder) but each layer contains a set of long convolutional filters. Second, a Dilated TCN uses dilated convolutions instead of pooling and upsampling and adds skip connections between layers. This

is an adaptation of the recent WaveNet [145] model, which shares similarities to our ED-TCN but was developed for speech processing tasks. The Dilated TCN has more layers, but each uses dilated filters that only operate on a small number of time steps. Empirically, we find both TCNs are capable of capturing features of segmental models, such as action durations and pairwise transitions between segments, as well as long-range temporal patterns similar to recurrent models. These models tend to outperform our previous methods, as well as a Bidirectional LSTM (Bi-LSTM) [146] baseline that is competitive with our previous approaches. One advantage of TCNs over Bi-LSTM is that they are over a magnitude faster to train. The ED-TCN in particular produces many fewer over-segmentation errors than other models.

Note that these models can be viewed as coming out of three different areas. First, they can be viewed as a hierarchical extension to the action primitives discussed throughout this thesis. Second, they overlap with recent ideas in Semantic Segmentation (e.g., [147, 148]). The Encoder-Decoder TCN is most similar to SegNet [147] whereas the Dilated TCN is most similar to the Multi-Scale Context model [148]. Lastly, TCNs are also related to Time-Delay Neural Networks (TDNNs), which were introduced by Waibel *et al.* [149] in the early 1990s. Note, TDNNs apply a hierarchy of temporal convolutions across the input but do not use pooling, skip connections, newer activations (e.g., Rectified Linear Units), or other features of our TCNs.

In this thesis we have focused on action segmentation, however, there are other papers that approach the problem of action detection (e.g., [24, 53, 150, 151]). Despite

effectively being the same problem, the temporal methods in segmentation papers tends to differ from detection papers, as do the metrics by which they are evaluated. The difference is as follows. Action segmentation methods predict what action is occurring at every frame in a video and detection methods output a sparse set action segments, where a segment is defined by a start time, end time, and class label. Note, however, that it is possible to convert between a given segmentation and set of detections by simply adding or removing null/background segments.

In this chapter, we evaluate on datasets designed for both tasks and propose a segmental F1 score, which we qualitatively find is more applicable to real-world concerns for both segmentation and detection than common metrics. We evaluate on MERL Shopping which was designed for action detection, Georgia Tech Egocentric Activities which was designed for action segmentation, as well as 50 Salads and JIGSAWS.

5.2 Related Work

For context, we include a brief overview of recent work on action detection, beyond what was described in Chapter 2, and draw connections between TCNs and models in other research areas.

Action Detection: Many fine-grained detection papers use sliding window-based detection methods on spatial or spatiotemporal features. Rohrbach *et al.* [49] used Dense Trajectories [9] and human pose features on the MPII Cooking dataset. At

each frame they evaluated a sliding SVM for many candidate segment lengths and performed non-maximal suppression to find a small set of action predictions. Ni *et al.* [150] used an object-centric feature representation, which iteratively parses object locations and spatial configurations, and applied it to the MPII Cooking and ICPR 2012 Kitchen datasets. Their approach used Dense Trajectory features as input into a sliding-window detection method with segment intervals of 30, 60, and 90 frames. Singh *et al.* [24] improved upon this by feeding per-frame CNN features into an LSTM model and applying a method analogous to non-maximal suppression to the LSTM output. We use Singh's proposed dataset, MERL Shopping, and show our approach outperforms their LSTM-based detection model. Recently, Richard *et al.* [151] introduced a segmental approach that incorporates a language model, which captures pairwise transitions between segments, and a duration model, which ensures that segments are of an appropriate length. In the experiments we show that our model is capable of capturing both of these components.

Action Segmentation: Segmentation papers tend to use temporal models that capture high-level patterns, for example RNNs or Conditional Random Fields (CRFs). The line of work by Fathi *et al.* [3, 50, 23] used a segmental model that captured object states at the start and end of each action (e.g., the appearance of bread before and after spreading jam). They applied their work to the Georgia Tech Egocentric Activities (GTEA) dataset, which we use in our experiments. Singh *et al.* [152] used an ensemble of CNNs to extract egocentric-specific features on the GTEA dataset but

did not use a high-level temporal model. Recall in Chapter 4 we introduced a spatiotemporal CNN with a constrained segmental model which we applied to 50 Salads. This model reduced the number of action over-segmentation errors by constraining the maximum number of candidate segments. We show our TCNs produce even fewer over-segmentation errors.

Other related models: There are parallels between TCNs and recent work on semantic segmentation, which uses Fully Convolutional CNNs to compute a per-pixel object labeling of a given image. The Encoder-Decoder TCN is most similar to SegNet [147] whereas the Dilated TCN is most similar to the Multi-Scale Context model [148]. TCNs are also related to Time-Delay Neural Networks (TDNNs), which were introduced by Waibel *et al.* [149] in the early 1990s. TDNNs apply a hierarchy of temporal convolutions across the input but do not use pooling, skip connections, newer activations (e.g., Rectified Linear Units), or other features of our TCNs.

5.3 Temporal Convolutional Networks

In this section we define two TCNs, each of which have the following properties: (1) computations are performed layer-wise, meaning every time-step is updated simultaneously, instead of updating sequentially per-frame, (2) convolutions are computed across time, and (3) predictions at each frame are a function of a fixed-length period of time, which is referred to as the receptive field. Our ED-TCN uses an encoder-decoder

architecture with temporal convolutions and the Dilated TCN, which is adapted from the WaveNet model, uses a deep series of dilated convolutions.

The input to a TCN will be a set of video features, such as those output from a spatial or spatiotemporal CNN, for each frame of a given video. Let $X_t \in \mathbb{R}^{F_0}$ be the input feature vector of length F_0 for time step t for $1 \leq t \leq T$. Note that the number of time steps T may vary for each video sequence. The action label for each frame is given by vector $Y_t \in \{0,1\}^C$, where C is the number of classes, such that the true class is 1 and all others are 0.

5.3.1 Encoder-Decoder TCN

Our encoder-decoder framework is depicted in Figure 5.1. The encoder consists of L layers denotes by $E^{(l)} \in \mathbb{R}^{F_l \times T_l}$ where F_l is the number of convolutional filters in a the l-th layer and T_l is the number of corresponding time steps. Each layer consists of temporal convolutions, a non-linear activation function, and max pooling across time.

We define the collection of filters in each layer as $W = \{W^{(i)}\}_{i=1}^{F_l}$ for $W^{(i)} \in \mathbb{R}^{d \times F_{l-1}}$ with a corresponding bias vector $b \in \mathbb{R}^{F_l}$. Given the signal from the previous layer, $E^{(l-1)}$, we compute activations $E^{(l)}$ with

$$E^{(l)} = max\_pooling(f(W * E^{(l-1)} + b)), \tag{5.1}$$

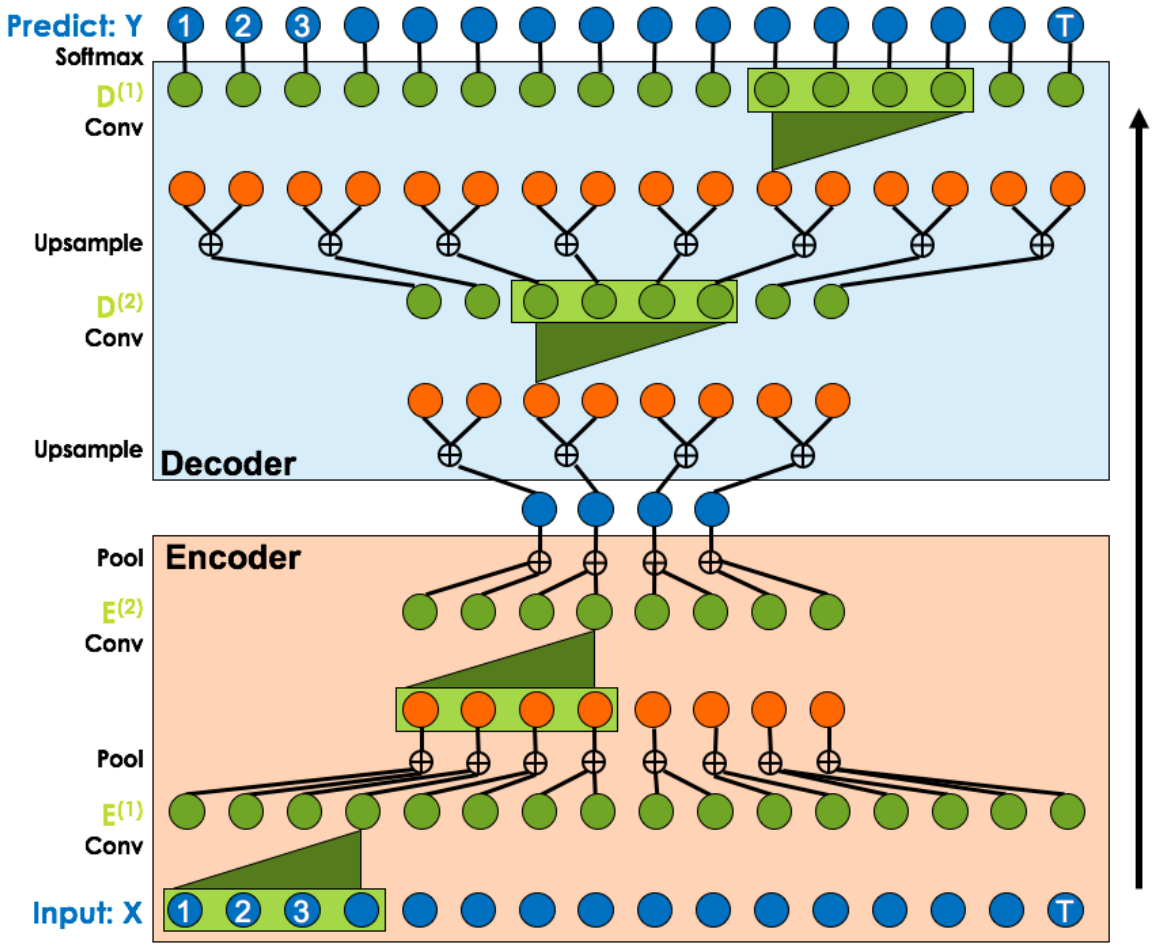


Figure 5.1: Our Encoder-Decoder Temporal Convolutional Network (ED-TCN) hierarchically models actions using temporal convolutions, pooling, and upsampling.

where $f(\cdot)$ is the activation function and $*$ is the convolution operator as defined in Equation 3.16.[1]

We compare activation functions in Section 5.4.4 and find normalized Rectified Linear Units perform best on the evaluated datasets. After each activation function, we max pool with width 2 across time such that $T_l = \frac{1}{2}T_{l-1}$. Pooling enables us to efficiently compute activations over long temporal windows.

Our decoder is similar to the encoder, except that upsampling is used instead of pooling and the order of the operations is now upsample, convolve, and apply the activation function. Upsampling is performed by simply repeating each entry twice. The convolutional filters in the decoder distribute the activations from the condensed layers in the middle to the action predictions at the top. Experimentally, these convolutions provide a large improvement in performance and appear to capture pairwise transitions between actions. Each decoder layer is denoted by $D^{(l)} \in \mathbb{R}^{F_l \times T_l}$ for $l \in \{L, \ldots, 1\}$. Note that these are indexed in reverse order compared to the encoder, so the filter count in the first encoder layer is the same as in the last decoder layer.

Vector $\hat{Y}_t \in [0,1]^C$ corresponds to the estimated probability of frame t being one of the C action classes. This is defined using weight matrix $U \in \mathbb{R}^{C \times F_1}$ and bias

[1]Note that $W * E^{(l-1)}$ is of size $F_l \times T$ and b is of length F_l. The addition operator here broadcasts each element of b across each row of the convolution output. Mathematically this is equivalent to $W * E^{(l-1)} + b\mathbf{1}_{T_l}$ where $\mathbf{1}_{T_l}^\top$ is a vector of length F_l and $\cdot^\top$ is the transpose operator.

$c \in \mathbb{R}^C$ with a softmax function, such that

$$\hat{Y}_t = \text{Softmax}(UD_t^{(1)} + c). \tag{5.2}$$

Furthermore, let $\hat{y}_t = \arg\max_c \hat{Y}_{t,c}$ be the index corresponding to the most probable class.

We explored other mechanisms, such as skip connections between layers, different patterns of convolutions, and other normalization schemes, however, the proposed model outperformed these alternatives and is arguably simpler. Implementation details are described in Section 5.3.3.

Receptive Field: The prediction at each frame is a function of a fixed-length period of time, which is given by the $r(d, L) = d(2^L - 1) + 1$ for L layers and duration d.

5.3.2 Dilated TCN

We adapt the WaveNet [145] model, which was designed for speech synthesis, to the task of action segmentation. In their work, the predicted output, $\hat{y}_t$, denoted which audio sample should come next given the audio from frames 1 to t. In our case $\hat{y}_t$ is the current action given the video features up to t.

As shown in Figure 5.2, we define a series of blocks, each of which contains a sequence of L convolutional layers. The activations in the l-th layer and j-th block are given by $S^{(j,l)} \in \mathbb{R}^{F_w \times T}$. Note that each layer has the same number of filters F_w.

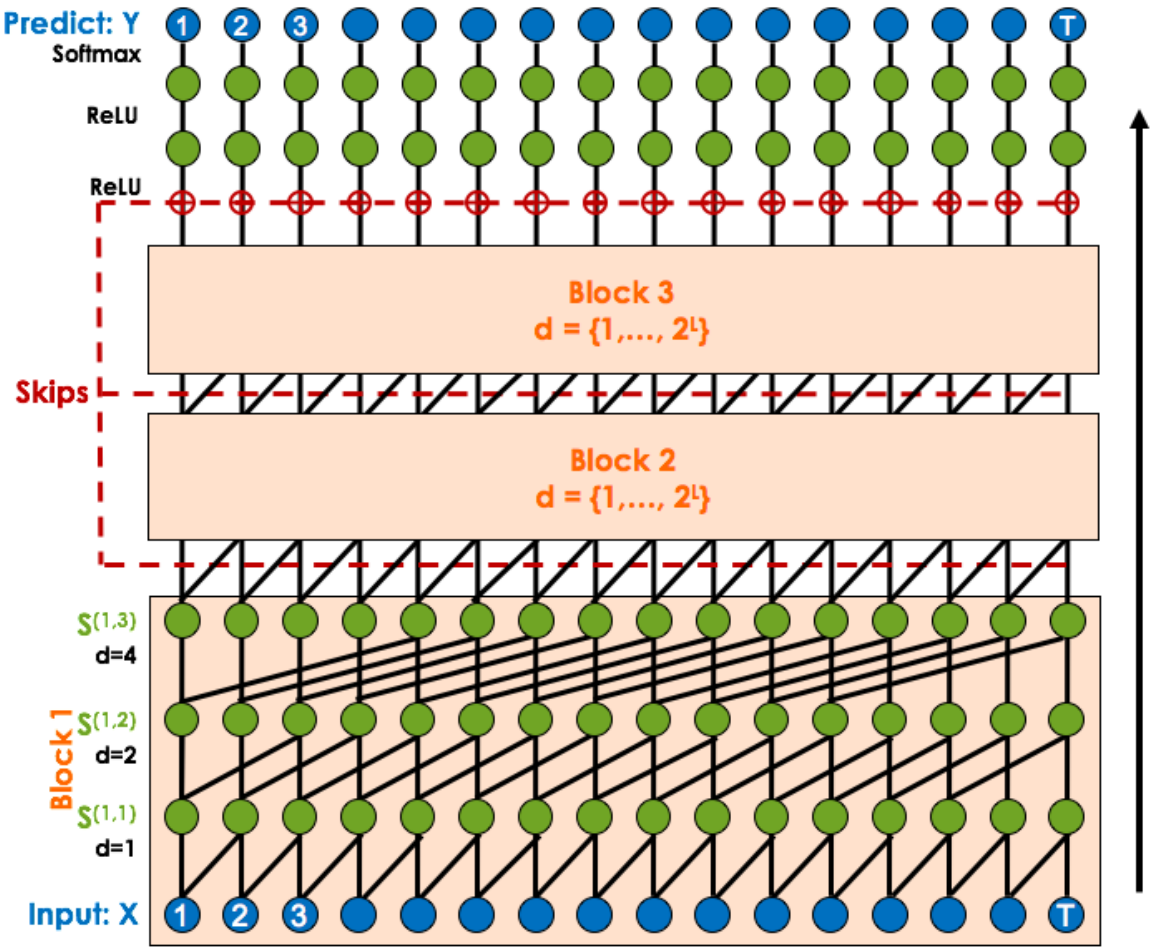


Figure 5.2: The Dilated TCN model uses a deep stack of dilated convolutions to capture long-range temporal patterns.

This enables us to combine activations from different layers later. Each layer consists a set of dilated convolutions with rate parameter s, a non-linear activation $f(\cdot)$, and a residual connection than combines the layer's input and the convolution signal. A dilated convolution is defined using convolution operator $*_s$ such that

$$W^{(i)} *_s E^{(l-1)} = \sum_{t'=1}^{d} \sum_{j=1}^{F_w} W_{t',j}^{(i)} E_{j,t-s*t'+1}^{(l-1)}. \tag{5.3}$$

Residuals connections were popularized by He *et al.* [136] and are defined as follows. For arbitrary input x and activation function $f(\cdot)$, the output of residual connection is simply $x + f(x)$. For image classification, this proves important for training deep networks with hundreds of layers [136], potentially because of its effect on the gradients computed during learning. This may also be useful in WaveNet due to the high sampling rate of audio data. In our preliminary experiments on video data, residual connections did not have a large impact on performance; however, we do use them to stay consistent with WaveNet.

The dilated convolutions in WaveNet are only applied over two time steps, t and $t - s$, so we simplify the dilated convolution equations below. The filters are parameterized by $W = \{W^{(1)}, W^{(2)}\}$ with $W^{(i)} \in \mathbb{R}^{F_w \times F_w}$ and bias vector $b \in \mathbb{R}^{F_w}$. As depicted in Figure 5.2, the input into each block $S^{(j,1)}$ is the output from the

previous block $S^{(j-1,L)}$, except for the first block which is defined as the input data:

$$S^{(j,1)} = \begin{cases} S^{(j-1,L)} & \text{if } j > 1 \\ X & \text{otherwise} \end{cases}. \tag{5.4}$$

Let $\hat{S}_t^{(j,l)}$ be the result of the dilated convolution at time t and $S_t^{(j,l)}$ be the result after adding the residual connection such that

$$\hat{S}_t^{(j,l)} = f(W^{(1)} S_{t-s_l}^{(j,l-1)} + W^{(2)} S_t^{(j,l-1)} + b) \tag{5.5}$$

$$S_t^{(j,l)} = S_t^{(j,l-1)} + V \hat{S}_t^{(j,l)} + e. \tag{5.6}$$

Let $V \in \mathbb{R}^{F_w \times F_w}$ and $e \in \mathbb{R}^{F_w}$ be a set of weights and biases for the residual. Note that parameters $\{W, b, V, e\}$ are separate for each layer; we do not indicate this in Equation 5.5 for notational clarity. The input to the network is either sensor or video data.

The dilation rate increases for consecutive layers within a block such that $s_l = 2^l$. This enables us to increase the receptive field by a substantial amount without drastically increasing the number of parameters.

The output of each block is summed using a set of skip connections with $Z^{(0)} \in$

$\mathbb{R}^{F_w \times T}$ such that

$$Z_t^{(0)} = \text{ReLU}(\sum_{j=1}^{B} S_t^{(j,L)}). \tag{5.7}$$

There is a set of latent states $Z_t^{(1)} = ReLU(V_r Z_t^{(0)} + e_r)$ for weight matrix $V_r \in \mathbb{R}^{F_w \times F_w}$ and bias e_r. The predictions for each time t are given by

$$\hat{Y}_t = \text{Softmax}(U Z_t^{(1)} + c) \tag{5.8}$$

for weight matrix $U \in \mathbb{R}^{C \times F_w}$ and bias $c \in \mathbb{R}^C$.

Receptive Field: The filters in each Dilated TCN layer are smaller than in ED-TCN, so in order to get an equal-sized receptive field it needs more layers or blocks. The expression for its receptive field is $r(B, L) = B * 2^L$ for number of blocks B and number of layers per block L.

5.3.3 Implementation Details

Parameters of both TCNs are learned using the categorical cross entropy loss with Stochastic Gradient Descent and ADAM [134] step updates. For ED-TCN, each of the L layers has $F_l = 96 + 32 * l$ filters. For the Dilated TCN we find that performance does not depend heavily on the number of filters for each convolutional layer, so we set $F_w = 128$. We perform ablative analysis with various number of layers and filter durations in the experiments. All models were implemented using Keras [135] and

TensorFlow [153].

We use dropout on full convolutional filters as proposed by Tompson *et al.* [154]. The way traditional dropout [155] works is as follows. For every training iteration, set a random percentage of the weights in a layer to zero. This is commonly just used in the fully connected layers of a CNN. When the traditional method is applied to convolutional weights, some of the weights for a given filter are set to zero but others retain their original value. This results in noisy-looking filters because only some of the weights are updated in each iteration. In the approach by Tompson *et al.*, instead of dropping out some of the weights in a filter, either all of the weights are set to zero or none of them are. This results in some percentage of full filters being zero for a given training iteration. We use this approach on our temporal convolutional filters and find it improves performance and produces smoother looking weights.

5.3.4 Causal versus Acausal

We perform causal and acausal experiments. Causal means that the prediction at time t is only a function of data from times 1 to t, which is important for applications in robotics. Acausal means that the predictions may be a function of data at any time step in the sequence. For the causal case in ED-TCN, for at each time step t and filter length d, we convolve from $t - d$ to t. In the acausal case we convolve from $t - \frac{d}{2}$ to $t + \frac{d}{2}$.

For the acausal Dilated TCN, we modify Eqn 5.5 such that each layer now operates

over one previous step, the current step, and one future step:

$$\hat{S}_t^{(j,l)} = f(W^{(1)} S_{t-s_l}^{(j,l-1)} + W^{(2)} S_t^{(j,l-1)} + W^{(3)} S_{t+s_l}^{(j,l-1)} + b) \tag{5.9}$$

where now $W = \{W^{(1)}, W^{(2)}, W^{(3)}\}$.

5.4 Evaluation

First we define our metrics. Then we perform synthetic experiments that highlight the ability of our TCNs to capture high-level temporal patterns. Finally we perform quantitative experiments on three challenging datasets and ablative analysis to measure the impact of hyper-parameters such as filter duration.

5.4.1 Metrics

Table 5.1 includes a list of recent fine-grained segmentation and detection datasets used by the computer vision community. Many of the datasets are evaluated on using different metrics in different papers or using different action granularities and label sets. For example, there have been at least three label sets on the Georgia Tech Egocentric Activities dataset consisting of 11, 62, or 72 action classes per set. Four research groups have used 50 Salads and each either added or completely replaced the original metrics.

In this chapter we evaluate using metrics from the segmentation and detection

Dataset	Problem	Metrics w/ papers
CMU Kitchen	Seg	Fr Acc [97, 41], Macro Acc [97]
ICPR Kitchen	Seg	F-score [53, 150] Fr+Seg Acc [92]
50 Salads	Seg	Fr Acc [112, 1], Fr P/R [12, 1], Seg Edit [112, 1], IoU mAP [151], Macro Acc [97]
GTEA	Seg	Fr Acc [3, 23, 50, 5, 92] Seg Acc [92]
MPII	Det	Midpoint mAP+PR [49, 151, 53, 150, 97]
ADL	Seg	Acc[92, 97]
MERL Shopping	Det	Midpoint mAP[156]

Table 5.1: Fine-grained action segmentation and detection datasets. Metrics abbreviations: Fr=Framewise, Acc=Accuracy, Seg=Segmental, P/R=Precision/Recall, mAP=Mean Average Precision.

communities and introduce a segmental F1 score, which can be applied in a meaningful way to either task. The appendix in Section 5.6 includes more detailed descriptions and equations for these metrics.

Segmentation metrics: Action segmentation papers tend use to frame-wise metrics, such as accuracy, precision, and recall [12, 157]. One drawback of many frame-wise approaches is that there may be large qualitative differences between several models that all achieve similar accuracy. As we described in Chapter 3, a model may achieve high accuracy but produce numerous over-segmentation errors. Nonetheless, for completeness we evaluate all methods using frame-wise accuracy. We visualize this issue in our results.

Detection metrics: Action detection papers tend to use segment-wise metrics such as mean Average Precision with midpoint hit criterion (mAP@mid) [49, 24] or mAP with a intersection over union (IoU) overlap criterion (mAP@k) [151]. mAP@k is

computed my comparing the overlap score for each segment with respect to the ground truth action of the same class. If an IoU score is above a threshold of k percent it is considered a true positive, otherwise it is a false positive. Average precision is computed for each class and the results are averaged. mAP@mid is similar except the criterion for a true positive is whether or not the midpoint (mean time) is within the start and stop time of the corresponding correct action.

mAP is a useful metric for information retrieval tasks like video search, however for many fine-grained action detection applications, such as robotics or video surveillance, we find that results are not indicative of real-world performance. The key issue is that mAP is very sensitive to a confidence score assigned to each segment prediction. These confidences are often simply the mean or maximum class score within the frames corresponding to a predicted segment. By computing these confidences in subtly different ways you obtain wildly different results. On MERL Shopping, the mAP@mid scores for Singh *et al.* [24] jump from 50.9 using the mean prediction score over an interval to 69.8 using the maximum score over that same interval.

F1@k: We propose a segmental F1 score which is applicable to both segmentation and detection tasks and has the following properties: (1) it penalizes over-segmentation errors, (2) it does not penalize for minor temporal shifts between the predictions and ground truth, which may have been caused by annotator variability, and (3) scores are dependent on the number actions and not on the duration of each action instance. This metric is similar to mAP with IoU thresholds except that it does

not require a confidence for each prediction. Qualitatively, we find these numbers are better at indicating the caliber of a given segmentation.

We compute whether or not each predicted action segment is a true or false positive by comparing its IoU with respect to the corresponding ground truth using threshold k. As with mAP detection scores, if there is more than one correct detection within the span of a single true action then only one is marked as a true positive and all others are false positives. We compute precision and recall for true positives, false positives, and false negatives summed over all classes and compute $F1 = 2\frac{prec*recall}{prec+recall}$.

We attempted to obtain action predictions from the original authors on all datasets to compare across multiple metrics. We received them for 50 Salads and MERL Shopping.

5.4.2 Synthetic Experiments

We claim TCNs are capable of capturing complex temporal patterns, such as action compositions, action durations, and long-range temporal dependencies. We show these abilities with two synthetic experiments. For each, we generate toy features X and corresponding labels Y for 50 training sequences and 10 test sequences of length $T = 150$. The duration of each action of a given class is fixed and action segments are sampled randomly. An example for the composition experiment is shown in Figure 5.3. Both TCNs are acausal and have a receptive field of length 16.

Action Compositions: By definition, an activity is composed of a sequence of

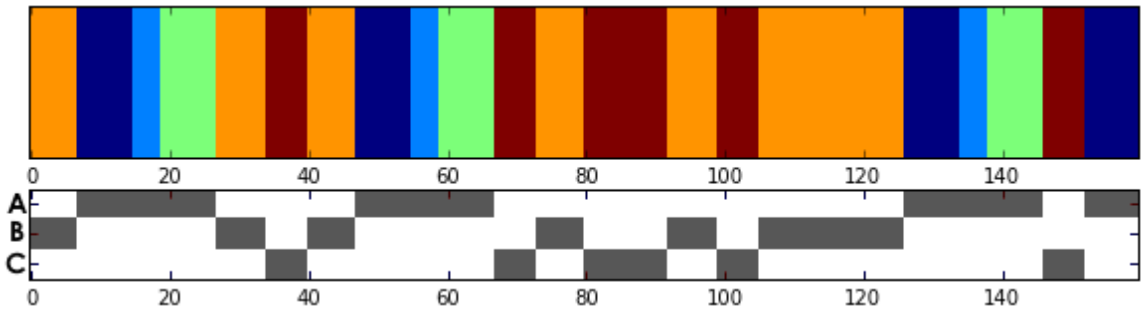


Figure 5.3: Synthetic Experiment #1: (top) True action labels for a given sequence (bottom) The 3 dimensional features for that sequence. White is -1 and gray is $+1$. Subactions A1, A2, and A3 (dark blue, light blue and green) all have the same feature values, which differ from B (orange) and C (red).

actions. Typically there is a dependency between consecutive actions (e.g., action B likely comes after A). CRFs capture this using a pairwise transition model over class labels and RNNs capture it using LSTM across latent states. We show that TCNs can capture action compositions, despite not explicitly conditioning the activations at time t on previous time steps within that layer.

In this experiment, we generated sequences using a Markov model with three high-level actions A, B, and C with subactions A_1, A_2, and A_3, as shown in Figure 5.3. A always consists of subactions A_1 (dark blue), A_2 (light blue), then A_3 (green), after which it is transitions to B or C. For simplicity, $X_t \in \mathbb{R}^3$ corresponds to the high-level action Y_t such that the true class is $+1$ and others are -1.

The feature vectors corresponding to subactions $A1 - A3$ are all the same, thus a simple frame-based classifier would not be able to differentiate them. The TCNs, given their long receptive fields, segment the actions perfectly. This suggests that our TCNs are capable of capturing action compositions. Recall each action class had a

Shift	s=0	s=5	s=10	s=15	s=20
ED-TCN	100	97.9	89.5	74.1	57.1
Dilated TCN	100	92.7	87.0	69.6	61.5
Bi-LSTM	100	72.3	60.2	54.7	38.5

Table 5.2: Synthetic experiment #2: F1@10 when shifting the input features in time with respect to the true labels. Column shows performance when shifting the data s frames from the corresponding labels. Each TCN receptive field is 16 frames.

different segment duration, and we correctly labeled all frames, which suggests TCNs can capture duration properties for each class.

Long-range temporal dependencies: For many actions it is important to consider information from seconds or even minutes in the past. For example, in the cooking scenario, when a user cuts a tomato, they tend to occlude the tomato with their hands, which makes it difficult to identify which object is being cut. It would be advantageous to recognize that the tomato is on the cutting board before the user starts the cutting action. In this experiment, we show TCNs are capable of learning these long-range temporal patterns by adding a phase delay to the features. Specifically, for both training and test features we define $\hat{X}$ as $\hat{X}_t = X_{t-s}$ for all t. Thus, there is a delay of s frames between the labels and the corresponding features.

Results using F1@10 are shown in 5.2. For comparison we show the TCNs as well as Bi-LSTM. As expected, with no delay ($s = 0$) all models achieve perfect prediction. For short delays ($s = 5$), TCNs correctly detect all actions except the first and last of a sequence. As the delay increases, ED-TCN and Dilated TCN perform very well up to about half the length of the receptive field. The results for Bi-LSTM degrade

at a much faster rate.

5.4.3 Datasets

Here we recap the MERL shopping and GTEA datasets, as introduced in Chapter 2. 50 Salads and JIGSAWS are used as described in Chapters 3 and 4. For the video results in 50 Salads and JIGSAWS we use the spatial CNN features from Chapter 4 as input into our TCNs.

MERL Shopping [24]: is an action detection dataset consisting of 106 surveillance-style videos in which users interact with items on store shelves. The camera viewpoint is fixed and only one user is present in each video. There are five actions plus a background class: `reach to shelf`, `retract hand from shelf`, `hand in shelf`, `inspect product`, `inspect shelf`. Actions are typically a few seconds long.

We use the features from Singh *et al.* [24] as input. Singh's model consists of four VGG-style spatial CNNs: one for RGB, one for optical flow, and ones for cropped versions of RGB and optical flow. We stack the four feature-types for each frame and use Principal Components Analysis with 50 components to reduce the dimensionality. The train, validation, and test splits are the same as described in [24]. Data is sampled at 2.5 frames/second.

Georgia Tech Egocentric Activities (GTEA) [3]: contains 28 videos of 7 kitchen activities such as making a sandwich and making coffee. The four subjects performed each activity once. The camera is mounted on the user's head and is pointing towards

their hands. On average there are about 19 (non-background) actions per video and videos are around a minute long. We use the 11 action classes defined in [23] and evaluate using leave one user out. Cross-validation is performed over users 1-3 as done by [152]. We use a sampling rate of 3 frames per second.

We were unable to obtain state-of-the-art features from [152], so we trained spatial CNNs from scratch using the spatial CNN described in Chapter 4. Results using this CNN are similar, but slightly lower, than EgoNet [152], which we also compare against. We use these CNN features as input in all of the temporal models.

5.4.4 Experimental Results

To make the baselines more competitive, we apply Bidirectional LSTM (Bi-LSTM) [146] to 50 Salads and GTEA. We use 64 latent states per LSTM direction with the same loss and learning methods as previously described. The input to this model is the same as for the TCNs. Note that MERL Shopping already has this baseline.

50 Salads: Results on both action granularities are included in Table 5.3. All methods are evaluated in acausal mode. ED-TCN outperforms all other models on both granularities and on all metrics. We also compare against Richard *et al.* [151] who evaluated on the mid-level and reported using IoU mAP detection metrics. Their approach achieved 37.9 mAP@10 and 22.9 mAP@50. The ED-TCN achieves 64.9 mAP@10 and 42.3 mAP@50 and Dilated TCN achieves 53.3 mAP@10 and 29.2 mAP@50.

50 Salads (eval)	**F1@{10, 25, 50}**	**Edit**	**Acc**
Spatial CNN [112]	35.0, 30.5, 22.7	25.5	68.0
Dilated TCN	55.8, 52.3, 44.3	46.9	71.1
ST-CNN [112]	61.7, 57.3, 47.2	52.8	71.3
Bi-LSTM	72.2, 68.4, 57.8	67.7	70.9
ED-TCN	**76.5**, **73.8**, **64.5**	**72.2**	**73.4**
50 Salads (mid)	**F1@{10, 25, 50}**	**Edit**	**Acc**
Spatial CNN [112]	32.3, 27.1, 18.9	24.8	54.9
IDT+LM [151]	44.4, 38.9, 27.8	45.8	48.7
Dilated TCN	52.2, 47.6, 37.4	43.1	59.3
ST-CNN [112]	55.9, 49.6, 37.1	45.9	59.4
Bi-LSTM	62.6, 58.3, 47.0	55.6	55.7
ED-TCN	**68.0**, **63.9**, **52.6**	**59.8**	**64.7**

Table 5.3: Video-based action segmentation results 50 Salads. F1@k is our segmental F1 score, Edit is the Segmental edit score from [1], and Acc is frame-wise accuracy.

MERL (acausal)	**F1@{10, 25, 50}**	**mAP**	**Acc**
MSN Det [24]	46.4, 42.6, 25.6	**81.9**	64.6
MSN Seg [24]	80.0, 78.3, 65.4	69.8	76.3
Dilated TCN	79.9, 78.0, 67.5	75.6	76.4
ED-TCN	**86.7**, **85.1**, **72.9**	74.4	**79.0**
MERL (causal)	**F1@{10, 25, 50}**	**mAP**	**Acc**
MSN Det [24]	-	**77.6**	-
Dilated TCN	72.7, 70.6, 56.5	72.2	73.0
ED-TCN	**82.1, 79.8, 64.0**	64.2	**74.1**

Table 5.4: MERL Shopping results. Action segmentation results on the MERL Shopping dataset. Causal only uses features from previous time steps and acausal uses previous and future time steps. mAP refers to mean Average Precision with midpoint hit criterion.

GTEA	**F1@{10,25,50}**	**Acc**
EgoNet+TDD [152]	-	**64.4**
Spatial CNN [112]	41.8, 36.0, 25.1	54.1
ST-CNN [112]	58.7, 54.4, 41.9	60.6
Dilated TCN	58.8, 52.2, 42.2	58.3
Bi-LSTM	66.5, 59.0, 43.6	55.5
ED-TCN	**72.2**, **69.3**, **56.0**	64.0

Table 5.5: Action segmentation results on the Georgia Tech Egocentric Activities dataset. F1@k is our segmental F1 score and Acc is frame-wise accuracy.

JIGSAWS

Sensor-based	**Edit**	**Acc**
LSTM [16]	75.3	80.5
Dilated TCN	75.6	79.2
LC-SC-CRF [1]	76.8	**83.4**
Bi-LSTM[16]	81.1	83.3
SD-SDL[126]	83.3	78.6
ED-TCN	**85.2**	82.5
Vision-based	**Edit**	**Acc**
MsM-CRF [33]	-	71.7
IDT [112]	8.5	53.9
VGG [112]	24.3	45.9
Spatial CNN	37.7	74.0
Dilated TCN	66.7	77.7
LC-SC-CRF	72.7	72.6
ST-CNN	74.7	81.0
ED-TCN	**86.0**	**81.6**

Table 5.6: Video and sensor results on JIGSAWS. All video models except MsM-CRF, IDT, and VGG use the same spatial CNN features as input.

Notice that ED-TCN, Dilated TCN, and ST-CNN all achieve similar frame-wise accuracy but very different F1@k and edit scores. ED-TCN tends to produce many fewer over-segmentations than competing methods. Figure 5.6 shows mid-level predictions for these models. Accuracy and F1 for each prediction is included for comparison.

Many errors on this dataset are due to the extreme similarity between actions and subtle differences in object appearance. For example, our models confuse actions using the vinegar and olive oil bottles, which have a similar appearance. Similarly, we confuse some cutting actions (e.g., `cut cucumber` versus `cut tomato`) and placing actions (e.g., `place cheese` versus `place lettuce`).

MERL Shopping: We compare against use two sets of predictions from Singh *et al.* [24], as shown in Table 5.4. The first, as reported in their paper, are a sparse set of action detections which are referred to as MSN Det. The second, obtained from the authors, are a set of dense (per-frame) action segmentations. The detections use activations from the dense segmentations with a non-maximal suppression detection algorithm to output a sparse set of segments. Their causal version uses LSTM on the dense activations and their acausal version uses Bidirectional LSTM.

While Singh's detections achieve very high midpoint mAP, the same predictions perform very poorly on the other metrics. As visualized in Figure 5.6 (right), the actions are very short and sparse. This is advantageous when optimizing for midpoint mAP, because performance only depends on the midpoint of a given action, however, it it not effective if you require the start and stop time of an activity. Interesting, this method does worst in F1 even for low overlaps.

As expected the acausal TCNs perform much better than the causal variants. This verifies that using future information is important for achieving best performance. In the causal and acausal results the Dilated TCN outperforms ED-TCN in midpoint mAP, however, the F1 scores are better for ED-TCN. This suggests the confidences for the Dilated TCN are more reliable than ED-TCN.

Georgia Tech Egocentric Activities: Performance of the ED-TCN is on par with the ensemble approach of Singh *et al.* [152], which combines EgoNet features with TDD [158]. Recall that Singh's approach does not incorporate a temporal model,

so we expect that combining their features with our ED-TCN would improve performance. Unlike EgoNet and TDD, our approach uses simpler spatial CNN features which can be computed in real-time.

JIGSAWS: Video-based performance of ED-TCN is comparable in accuracy and slightly better than the sensor-based results. This is a significant jump from previous models, where sensor-based results far outperformed video results. Most surgical training tasks are not performed using robots or with complex sensing so this result is of important practical value.

Within the sensor-based results, the ED-TCN achieves similar accuracy compared to the LC-SC-CRF and Bi-LSTM models and outperforms all others in edit score. Note that three models – ED-TCN, LC-SC-CRF, and Bi-LSTM – each achieve practically the same accuracy, but impose very different assumptions on the data. This implies that 83% may be an upper limit on performance.

Figure 5.4 shows example predictions using ED-TCN on the video from JIGSAWS. These predictions were randomly sampled from the test sets to prevent cherry-picking good results. Note that the gold-colored actions in the 4th and 5th sequences from the top are rare and only appear in the trials from two users.

5.4.4.1 Ablative Experiments

Ablative experiments were performed on 50 Salads (mid-level). Note that these were done with different hyper-parameters – as defined below – and thus may not match

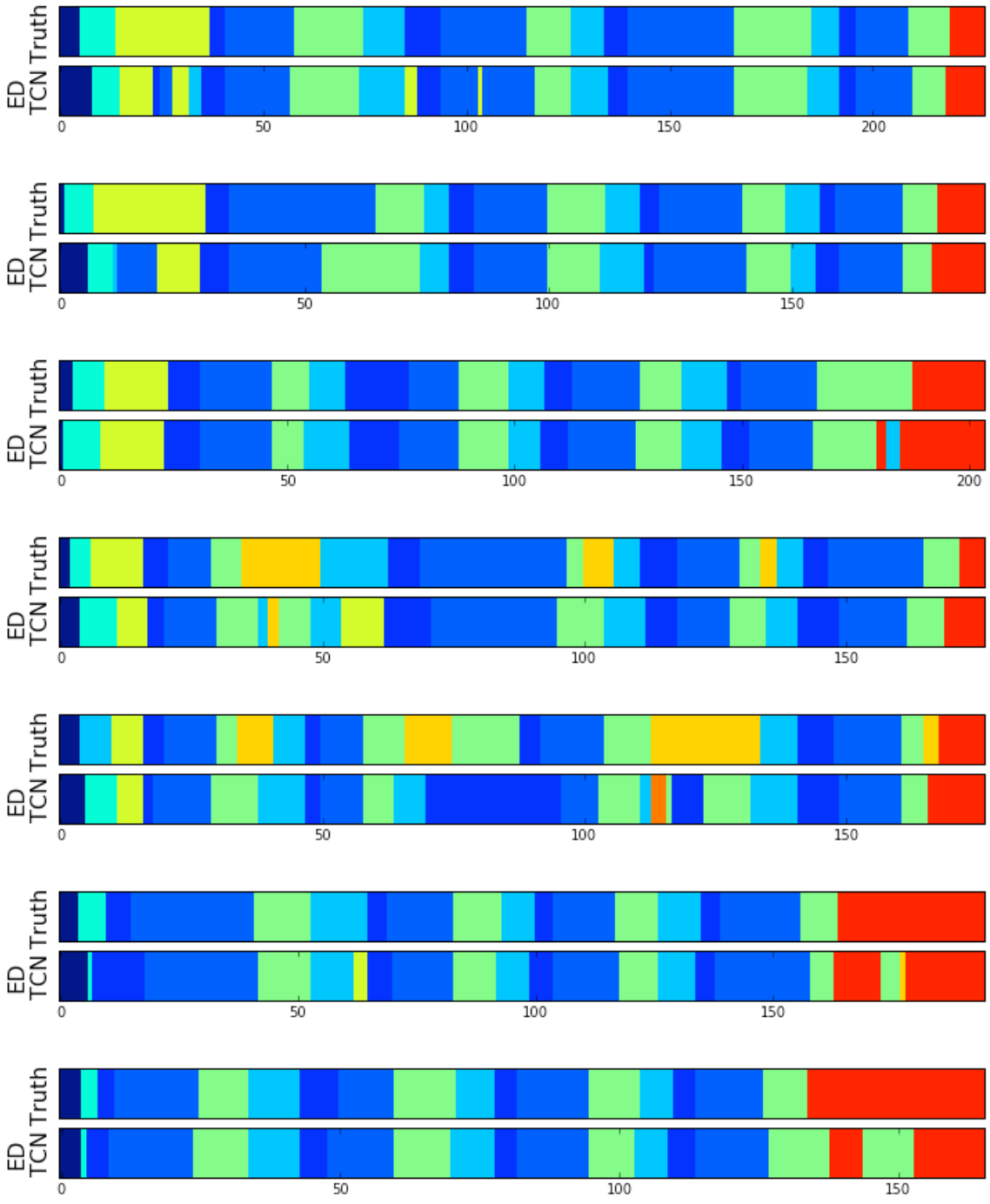


Figure 5.4: Timelines from ED-TCN on JIGSAWS (video).

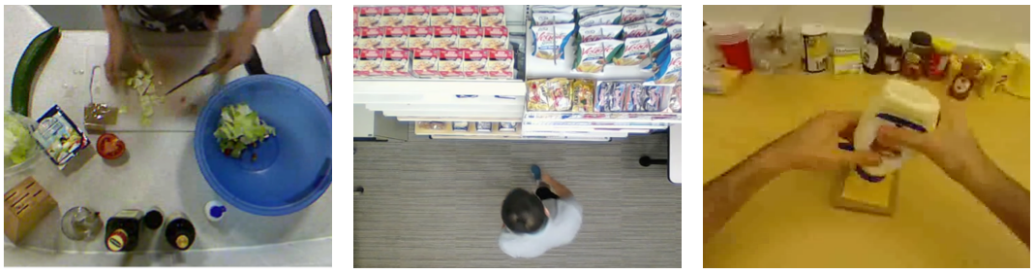


Figure 5.5: Example images from each dataset. (left) 50 Salads (center) MERL Shopping (right) GTEA.

Activation	Sigm.	ReLU	Tanh	GPC	NReLU
ED-TCN	37.3	40.4	48.1	52.7	**58.4**
Dilated TCN	42.5	**43.1**	41.0	40.5	40.7

Table 5.7: Comparison of different activation functions used in each TCN. Results are computed on 50 Salads (mid-level) with F1@25.

previous results.

Activation functions: We assess performance using the activation functions shown in Table 5.7. The Gated PixelCNN (GPC) activation [159], $f(x) = tanh(x) \odot sigmoid(x)$, was used for WaveNet and also achieves high performance on our tasks. We define the Normalized ReLU

$$f(x) = \frac{ReLU(x)}{\max(ReLU(x)) + \epsilon}, \tag{5.10}$$

for vector x and $\epsilon = 1\text{E-}5$ where the max is computed per-frame. Normalized ReLU outperforms all others with ED-TCN, whereas for Dilated TCN all functions are similar.

Receptive fields: We compare performance with varying receptive field hyperpa-

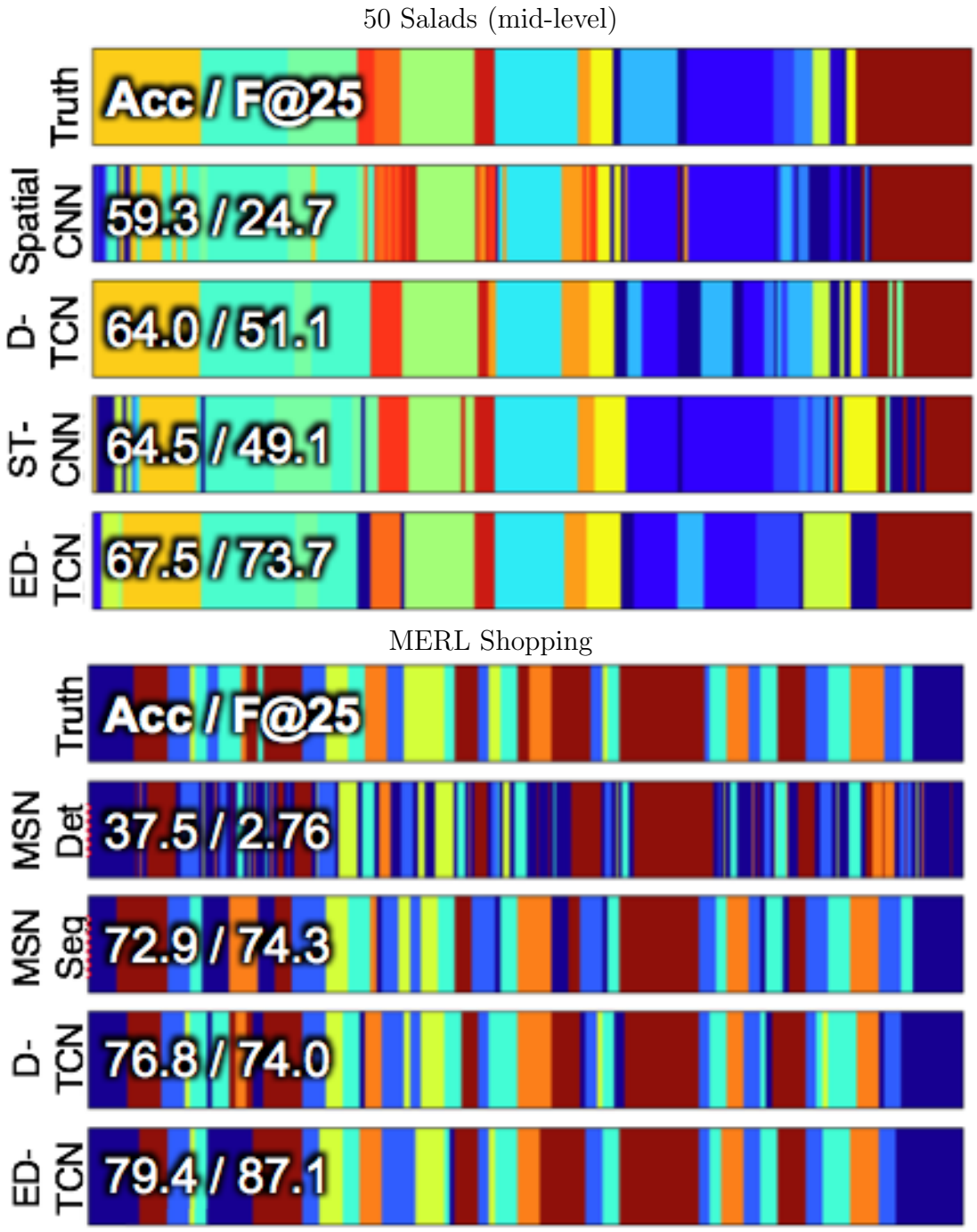


Figure 5.6: Action predictions for one sequence using the mid-level action set of 50 Salads (top) and on MERL Shopping (bottom). These timelines are "typical." Performance is near the average performance across each dataset. In each timeline "Acc" refers to the per-frame accuracy and F@25 refers to the F@k metric with an overlap of 25%.

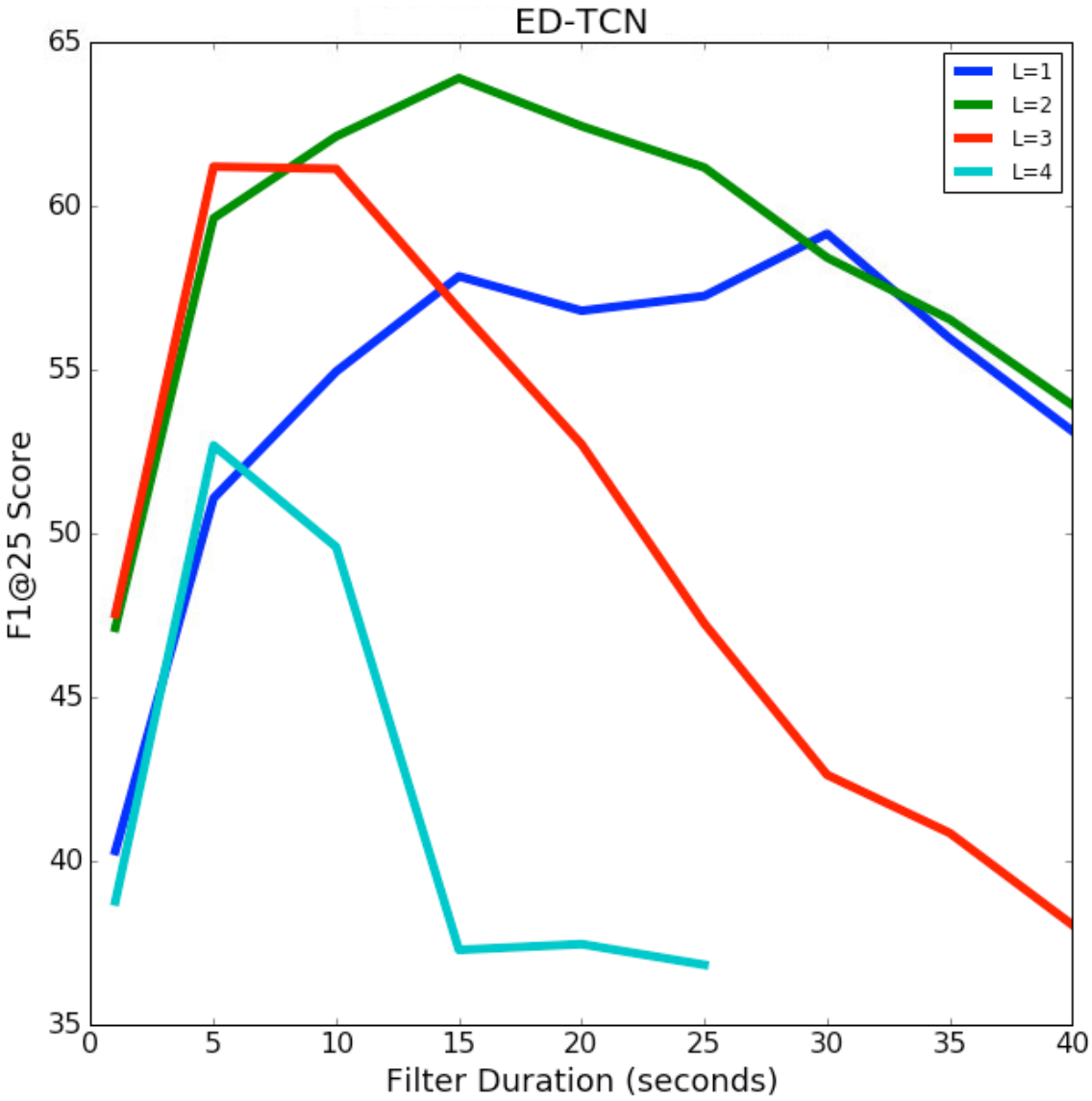


Figure 5.7: Receptive field experiments (ED-TCN): varying layer count L and filter durations d (right) Dilated TCN: varying layer count L and number of blocks B.

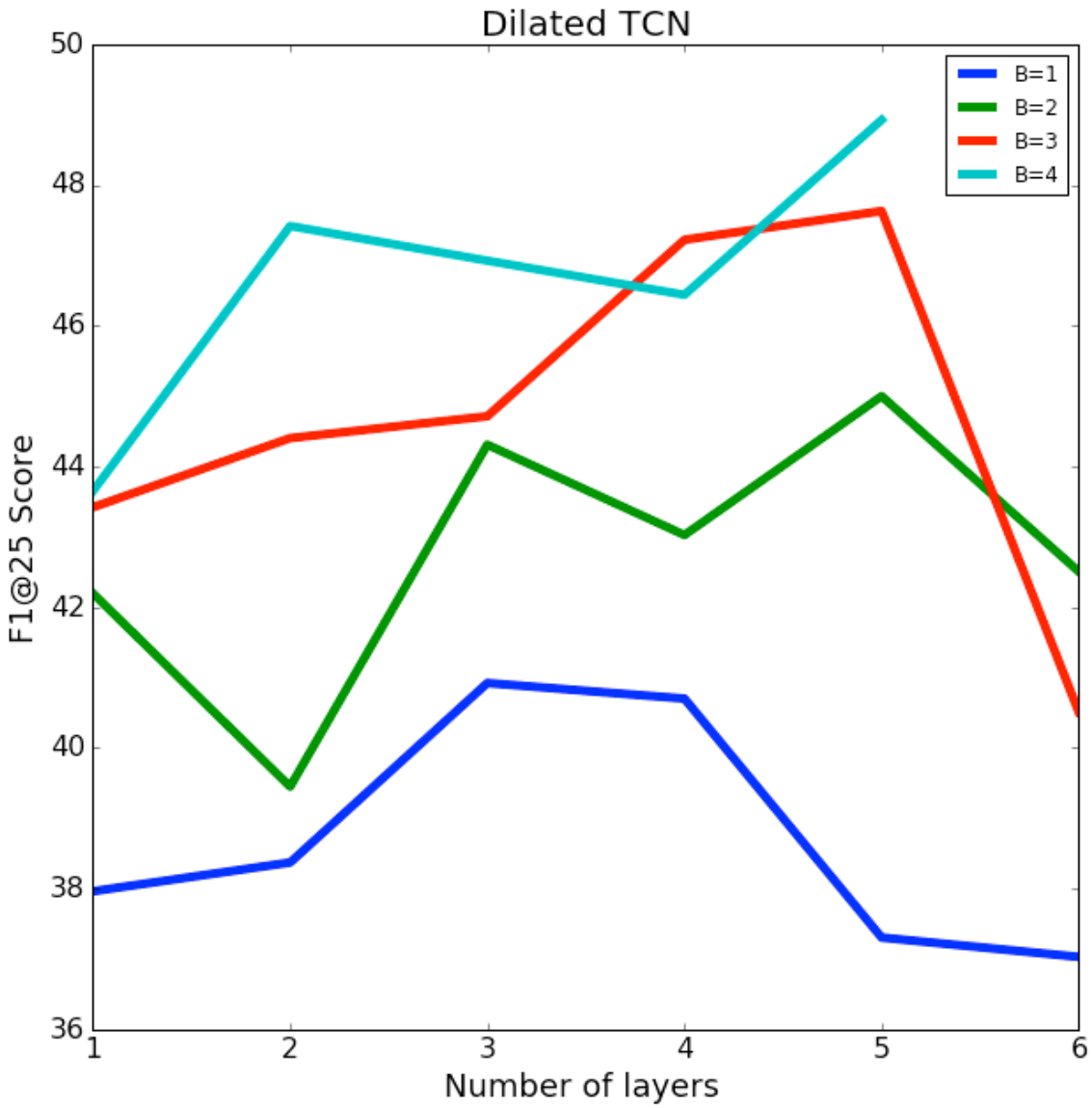


Figure 5.8: Receptive field experiments (Dilated TCN): varying layer count L and number of blocks B.

rameters. Line in Figure 5.7 show F1@25 for L from 1 to 5 and filter sizes d from 1 to 40 on ED-TCN. Lines in Figure 5.8 correspond to block count B with layer count L from 1 to 6 for a Dilated TCN. Note, our GPU ran out of memory on ED-TCN after ($L = 4$,$d = 25$) and Dilated TCN after ($B = 4$,$L = 5$). The ED-TCN performs best with a receptive field of 44 frames ($L = 2$,$d = 15$) which corresponds to 52 seconds. The Dilated TCN performs best at 128 frames ($B = 4$,$L = 5$) and achieves similar performance at 96 frames ($B = 3$,$L = 5$).

Training time: It takes much less time to train a TCN than a Bi-LSTM. While the exact timings vary with the number of TCN layers and filter lengths, for one split of 50 Salads – using a Nvidia Titan X for 200 epochs – it takes about a minute to train the ED-TCN whereas and 30 minutes to train the Bi-LSTM. This speedup comes from the fact that activations within each TCN layer are all independent, and thus they can be performed in batch on a GPU. Activations in intermediate RNN layers depend on previous activations within that layer, so operations must be applied sequentially.

5.5 Conclusion

We introduced Temporal Convolutional Networks, which use a hierarchy of convolutions to capture long-range temporal patterns. We showed on synthetic data that TCNs are capable of capturing complex patterns such as compositions, action du-

rations, and are robust to time-delays. Our models outperformed strong baselines, including Bidirectional LSTM, and achieved state of the art performance on challenging datasets. We believe TCNs are a formidable alternative to RNNs and are worth further exploration.

Overall, in our experiments the Encoder-Decoder TCN outperformed all other models, including state-of-the-art approaches for most datasets and our adaptation of the recent WaveNet model. The most important difference between these models is that ED-TCN uses fewer layers but has longer convolutional filters whereas the Dilated TCN has more layers but with shorter filters. The long filters in ED-TCN have a strong positive affect on F1 performance, in particular because they prevent over-segmentation issues. The Dilated TCN performs well on metrics like accuracy, but is less robust to over-segmentation.

5.6 Appendix: Metrics

For all metrics let y_t be the predicted label at time t and y_t^* be the corresponding true label.

Frame-wise Metrics: As the name implies, frame-wise metrics are evaluated independently for each frame and do not explicitly account for a segment's start or stop time. The two common frame-wise accuracy metrics are micro and macro. **Micro**

accuracy is simply the percentage of correctly predicted frames in a sequence.

$$A_{micro}(y, y^*) = \frac{1}{N} \sum_{i=1}^{N} \delta(y_i, y_i^*) \tag{5.11}$$

where δ is 1 if y_i and y_i^* are the same label and 0 otherwise.

Macro accuracy is the average accuracy per-class. This is computed by first computing the percentage of correct frames for each given class, and then computing the average of those accuracies:

$$A_{macro}(y, y^*) = \frac{1}{C} \sum_{c=1}^{C} \frac{1}{N} \sum_{i=1}^{N} \delta(y_i, y_i^*) \delta(y_i, c) \tag{5.12}$$

Frame-wise **precision and recall** are typically evaluated as macro metrics. For convenience we define true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). Precision refers to the percentage of correct predictions for class c compared to the total number of times a c was predicted:

$$Pr = \frac{TP}{TP + FP} \tag{5.13}$$

Recall is the percentage of correct predictions for class c compared to the number of correct true predictions or false negatives:

$$Re = \frac{TP}{TP + FN} \tag{5.14}$$

In other domains, like image classification, it is common to plot ROC curves, which show the trade off between precision and recall. However, this is not common for action segmentation.

The ICPR Kitchen dataset evaluates using an **F1 Score** which is the harmonic mean of the precision and recall for each class

$$F1 = \frac{2 \cdot Pr \cdot Pr}{Pr + Re}. \tag{5.15}$$

The frame-wise F1 score for each non-background class is then averaged.

Segmental Metrics: A segmental edit score measures how well the model predicts the ordering of action segments independent of temporal shifts. As described in Chapter 3, this segmental edit score is defined using a normalized edit distance, $s_e(G', P')$, with insertions, deletions, and replacements where G' and P' are the ground truth and predicted segment classes. The score is normalized by taking the maximum length of G' and P'.

Note that this metric is similar to Word Error Rate (WER) in natural language processing [160]. In the common NLP definition, the metric is often normalized by the length of the predicted sequence, and thus it is not guaranteed that the error will be bounded by 100%.

Our modified Jaccard **overlap score** measures overlap between ground truth and predicted segments and penalizes over-segmentation errors. This score is a function

of the longest contiguous predicted segment for a given ground truth segment. Let G be the ground truth labeling indexed by G_i for the ith segment from 1 to N and let P_i be the predicted labeling. The score is:

$$s_o(G, P) = \frac{100}{N} \sum_{i=1}^{N} \max_j \frac{|G_i \cap P_j|}{|G_i \cup P_j|} \tag{5.16}$$

It is similar to the Jaccard Index except ours penalizes over-segmentation errors.

Classification Accuracy: This metric assumes that we know the start time and duration of each action but not the action label. For prediction G, we compute the most likely action within the given segment interval and compute the segment-wise micro accuracy. This is often be treated as an upper bound on the accuracy expected without known segmentation.

Action Detection Metrics

Both detection metrics we discuss are used in tandem with mean Average Precision (mAP). Average Precision (AP) is defined as

$$AP = \int_{r=0}^{1} Pr(r)dr \approx \sum_{n=1}^{N} Pr(n)(Re(n) - Re(n-1)) \tag{5.17}$$

where $Pr(n)$ and $Re(n)$ are the precision and recall for the n-th prediction and N is the number of predicted segments. Precision and recall are computed as previously mentioned where $Pr(n)$ and $Re(n)$ are a function of all samples from 1 to N. mAP

is computed by averaging the AP for each non-background class.

For mAP, the order in which we rank segments is important. Thus, it is advantageous to process segments that we are most confident about before segments that we are less confident about. As such, we assume that for each segment we have a confidence, $c_i \in \mathbf{R}$, for the i-th segment. Segments are ordered from highest to lowest before computing the mAP.

Midpoint Hit Criterion: Midpoint Hit Criterion is the most common metric for action detection, as made popular through the MPII Cooking dataset [49]. The basic premise is simple: if the midpoint of a predicted segment lies between the start and end of the true segment, then it may be considered a true positive, otherwise it is a false positive. Furthermore, only one detection – the one with the highest confidence – may be considered a true positive. If there are multiple predictions of class c within a given (true) segment, then all predictions after the first are false positives. This penalizes over-segmentation issues. Midpoint hit criterion is used in tandem with mean average precision (mAP).

Intersection Over Union Thresholding: Intersection Over Union (IoU) thresholding is similar to the midpoint hit criterion, except that for each predicted segment you compute an overlap score instead of a "hit" or "miss". The first segment that has an overlap score above a specified threshold, $\tau \in [0, 1]$, is considered a true positive,

and all others are considered false positives. The IoU score is:

$$IoU(G, P) = \frac{|G_i \cap P_j|}{|G_i \cup P_j|} \tag{5.18}$$

for true segment G and predicted segment P. The mean average precision is then computed the same was as before. Typically results are shown for multiple thresholds (e.g., $\tau \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$).

Chapter 6

Surgical Phase Recognition: From Instrumented ORs to Hospitals Worldwide

Using the models previously described, we advance on recent work for video-based surgical phase recognition from intraoperative laparoscopic video. The goal is to segment sequence of phases, such as dissection, cutting, and hemostasis, from videos of live surgical procedures. Current methods for workflow analysis require extensive instrumentation on the tools, human annotations, or camera rigging within an operating room. We focus on video and multi-modal analysis using the methods we developed throughout this thesis. We perform comparative analysis of our models

on the TUM EndoVis dataset, which was collected from procedures in a single hospital, the M2CAI 2016 dataset which was collected from two university hospitals, and introduce EndoTube, our new dataset containing Cholecystectomy videos from over a dozen hospitals around the world. We show that despite high performance in constrained settings (e.g, EndoVis), current approaches are insufficient at capturing the extreme amounts of variability in more diverse environments.

Note, much of this work preceded development of our TCNs. We focus on our spatiotemporal models and include some updated TCN results at the end.

6.1 Introduction

Walk into an operating room for laparoscopic surgery and you will see a plethora of devices that can be instrumented and used for automatic workflow recognition. When available, these can be used to recognize surgical events, which may improve operating room efficiency [161], reduce information overload for surgeons [162], or retrospectively analyze surgical workflow [115]. However, most operating rooms do not have these devices or do not have a way of recording the data. In this work we address surgical phase recognition from laparoscopic video which is easy to collect in most ORs. In particular, we focus on offline (acausal) solutions for large-scale workflow analysis that can be performed across multiple institutions.

Recognizing surgical workflow from video is difficult due to large variability between patients, surgeons, and hospital environments. Patients exhibit substantial

variation in appearance due to differences in anatomy such as varying levels of fatty tissue. Surgeons tend to have their own style and may perform surgical phases in different temporal orders. Equipment, such as the endoscope and instruments, may be unique across hospitals, and can result in varying lighting conditions, video quality, and tool appearance. To model these elements of variability, we decompose surgical phase segmentation into two tasks: (1) learn a low-level spatiotemporal model that captures how the scene changes within short time intervals and (2) learn a high-level classifier that captures phase ordering.

Low-level: Individual surgical phases may be ambiguous but are often defined by the configurations of objects (e.g. tools, organs), their spatial relationships, and their motions throughout a sequence. We compare multiple CNN architectures including our spatial and spatiotemporal CNNs and AlexNet-like architectures as evaluated by [2]. We find that the spatiotemporal CNN offers large performance gains compared to the others. Recall that this model factorizes video into a spatial component that captures objects in a scene and a local temporal component that captures how these objects change over a short period of time (e.g. 60 seconds). For example, during the *clipping* phase the ST-CNN may capture the applicator tool motion as it applies a clip to the artery. One advantage, compared to the spatial-only models is that it explicitly encodes temporal information within the CNN.

High-level: We compare performance of the spatial and spatiotemporal CNN in tandem with the temporal classifiers we developed earlier to investigate the importance

of high-level temporal information, such as sequential phases ordering. Interestingly on small datasets such as EndoVis, the simple Dynamic Time Warping baseline from Chapter 3 outperforms all other models, however, on larger datasets, including M2CAI2016 and EndoTube, our classifiers perform better.

The topic of video-based surgical phase recognition from intraoperative laparoscopic video across many institutions has not been studied in the literature but is important for many applications such as retrospective skills assessment [163], workflow analysis [115], and surgical training [164]. Recent datasets, like EndoVis [165] and Cholec80 [2], have been collected from individual hospitals and do not capture the amount of variability seen across institutions. Factors like tool appearance and recording equipment may vary significantly between hospitals and cause video-based models to fail. We introduce a new dataset, EndoTube, which consists of 25 cholecystectomy videos from nine countries and over a dozen hospitals. These videos were carefully curated from procedures uploaded publicly by clinicians. In order to compare with current work, we use the same labels as EndoVis as shown in Figure 6.1. While performance on this dataset is significantly lower than EndoVis, we highlight the challenges of surgical data captured "in the wild" where there are many more types of variability. We also show results on the recent M2CAI 2016 challenge dataset.

Our primary contributions are: (1) exploring the use of spatial and spatiotemporal CNNs for representing surgical phases, (2) comparing several classifiers for capturing high level temporal information, and (3) performing analysis on EndoTube, our new

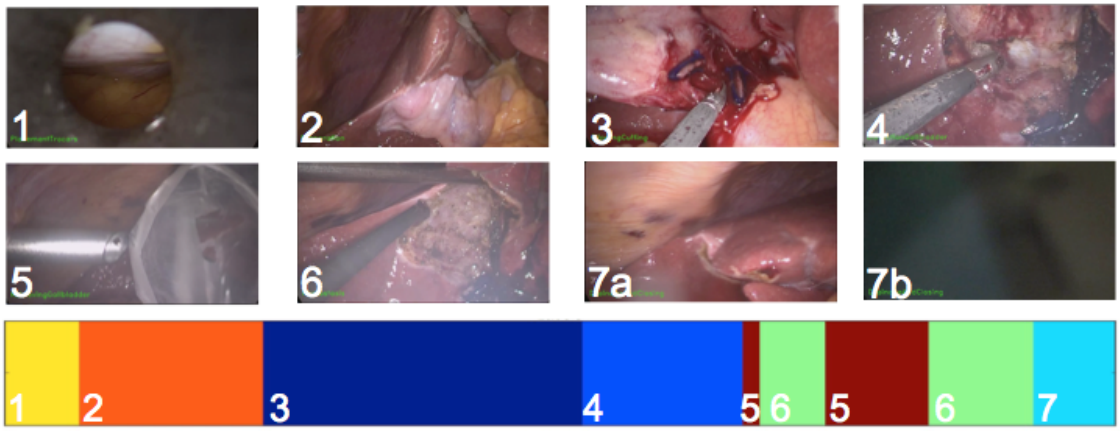


Figure 6.1: Example images and sequence labeling from the EndoVis dataset. Phases: (1) Place trocars (2) Prepare Calots triangle (3) Clip/cut cystic artery and duct (4) Dissect Gallbladder (5) Retrieve Gallbladder (6) Hemostasis (7a/7b) Drainage/closure/finish.

multi-institutional Cholecystetomy dataset.

6.2 Prior Work

Despite being a nascent area, there has been substantial recent interest in automated surgical workflow analysis due to publicly available data, promising initial results, and new methods from the computer vision community that may be more capable of modeling complex surgical video.

Recent work by Twinanda *et al.* [2] proposed a CNN-based approach to surgical phase recognition with a Hierarchical Hidden Markov Model. They achieved reasonable performance on the (public) EndoVis and (private) Cholec80 datasets, however, their best results on EndoVis required pre-training on a much larger surgical dataset. Whereas their CNN only captures spatial information per-image, our CNN explicitly

captures spatiotemporal information. Dergachyova *et al.* [166] showed high performance on EndoVis when combining video and tool data with a Hidden Semi-Markov Model approach, but achieved relatively low accuracy with their video-only variant compared to [2]. Their approach used hand-crafted image features like color histograms, Histogram of Oriented Gradients (HOG), and Local Binary Patterns (LBP). In both [2] and [166], performance using tool information is relatively low compared to video. Our bets model achieves superior performance using video, tools, and when these modalities are combined.

Lalys *et al.* [167] explored video-based phase recognition for pituitary surgery using microscope images. Their model does not use any temporal model which often results in gratuitous over-segmentation, and thus may not be applicable to applications like surgical summarization which require large coherent segments. Also related is the work by Padoy *et al.* [168] who instrumented an operating room with ceiling-mounted cameras and captured human motions to recognize surgical phases. They modeled the procedure as a Workflow Hidden Markov Model and used 3D motion flow features.

Earlier work showed that using auxiliary data like tool usage can be effective for workflow analysis [115, 169, 21], however, this requires recording and synchronizing tool data for each surgery which, at scale, is costly and cumbersome. In addition, it is difficult to collect this kind of data from multiple institutions. Padoy *et al.* [115] recognized surgical phases using hand-labeled tool usage information where they compared performance using an Annotated Hidden Markov Model and Dynamic Time

Warping. Stauder *et al.* [169] used a Random Forest-based approach to recognizing surgical phases from RFID tags attached to surgical instruments. Related work by Franke *et al.* [161] worked towards high-level planning of OR logistics modeled the time-remaining in a surgery by using generalized Surgical Process Models.

Much of the aforementioned work was applied to Cholecystectomies from individual institutions. Another key difference in our work is that we investigate how well current CNN-based models generalize to new hospitals from around the world with very different equipment and working conditions. While the surgical phases tend to be in a similar order between procedures within the same hospital, we find that there is larger variance when comparing against procedures at other hospitals. We also have work looking at generalizing more complicated procedures, which we do not describe here [21], but in this case all trials were all collected at the same hospital. There has also been work on identifying surgical phases for Cataracts surgery [170].

6.3 Methods

Our model is comprised of two components: first, we learn a spatiotemporal feature representation using a Convolutional Neural Network that encodes contextual information like tools, organs, and fluids, and models how they change over time. Second, we build a classifier that takes the spatiotemporal features as input and classifies surgical phases. These models were all described earlier in this thesis. We briefly summarize them here.

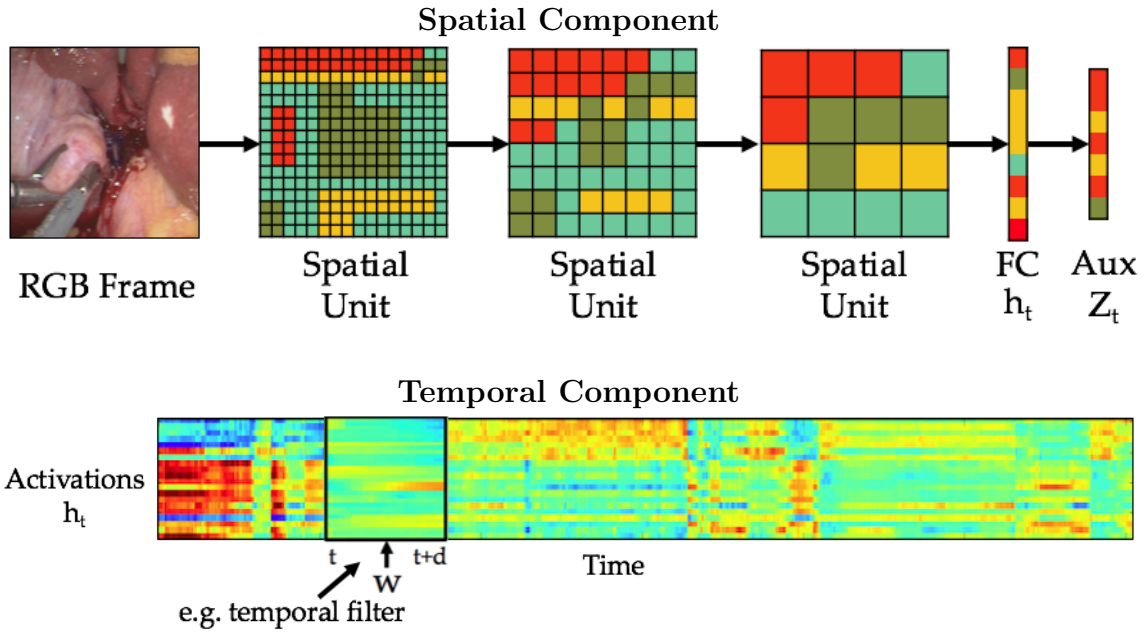


Figure 6.2: The Spatiotemporal CNN is factorized into spatial and temporal components. (top) The spatial component consists of spatial units that model the content in each region of an image. (bottom) The temporal component uses the spatial activations, h_t, as input and convolves a set of learned temporal filters. The output is a set of activations, s_t, that encode spatiotemporal information.

6.3.1 Spatiotemporal Video Representation

Let I_t be an RGB image for time t from 1 to T, $Z_t \in \{0,1\}^z$ be vector of z auxiliary signals, and $y_t \in \{1,\ldots,C\}$ be a phase label. The auxiliary signals can be tool usage information or phase labels as we describe later and the C phase labels are listed in Section 6.4. Given input image I_t, we compute spatiotemporal activations $s_t \in \mathbb{R}^p$ which is a vector of p latent states.

Spatial Component: The spatial component takes image I_t and outputs an intermediate spatial representation $h_t \in \mathbb{R}^f$. This model is composed of a set of spatial

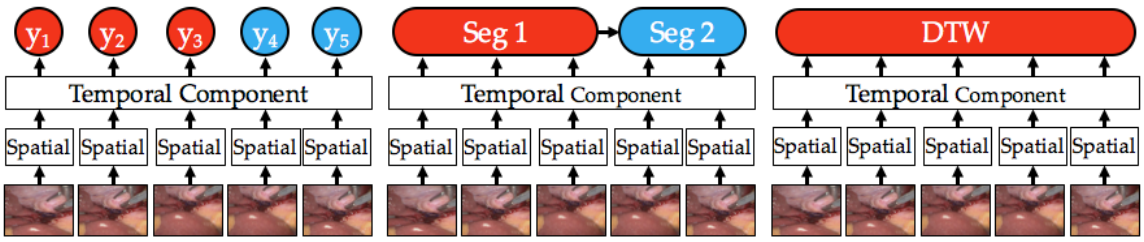


Figure 6.3: The full models with the Spatiotemporal CNNs and classifiers. (left) Linear Model (middle) Segmental Model (right) Time-invariant Model.

units and fully connected layers as shown in Figure 6.2 (top). The spatial units use convolutional filters to hierarchically model the content in each region of an image. Each of our three spatial units consists of a 3×3 convolutional layer, ReLU activation, and 3×3 max pooling. Each colored block in our depiction of a spatial unit corresponds to an activation vector in that region of the image.

The fully connected layer (FC) consists of latent states, each of which captures correlations between each region and the corresponding activations in that region. For example, a state may capture the tool being in the top right of the image and the gall bladder being in the middle. Let there be f states in each fully connected layer $h_t \in \mathbb{R}^f$ which correspond to different scene configurations.

The spatial component is trained using auxiliary data Z_t, weight vector $W^{(1)}$ and bias $b^{(1)}$ such that

$$\hat{Z}_t = g_{sp}(W^{(1)} h_t + b^{(1)}) \tag{6.1}$$

where $\hat{Z}_t$ is the predicted auxiliary signal. Early on in this work we performed ex-

periments using tool information or image attribute information as input Z. These experiments were similar to what we did in Chapter 4 where we trained our model with sensor data. Results were promising but not as good as using the true surgical phase labels. In this chapter we will assume Z_t is the surgical phase label for a given time step. As such, $g_{sp}(\cdot)$ is defined as the softmax function.

Temporal Component: Given the scene activations h_t the temporal component computes a set of temporal activations s_t. We learn temporal convolutional filters that capture how the spatial information changes over time.

Each of the l filters, $W_l^{(2)} \in \mathbb{R}^{d \times f}$, is convolved along time with the input, where d is the duration of a filter, $b_l^{(2)}$ is the bias for each filter:

$$s_t^l = \text{ReLU}(\sum_{t'=t}^{t+d-1} W_{l,t'}^{\top(2)} h_{t'} + b_l^{(2)}) \tag{6.2}$$

The temporal units are trained using phase labels $Y_t \in \{0,1\}^c$, where the index of the true class is 1 and all other classes are 0, using the softmax function with output weights and biases $W^{(3)}$ and $b^{(3)}$:

$$\hat{Y}_t = \text{softmax}(W^{(3)} s_t + b^{(3)}) \tag{6.3}$$

Predictions $\hat{Y}_t$ correspond to the predicted classes. Note that the input to each classifier is the spatiotemporal activations s_t for all time steps.

Implementation Details: Our network is trained using the cross entropy loss func-

tion with ADAM [134]. The three spatial units have $[32, 64, 96]$ filters in the convolutional layers, the first fully connected layer has $f = 128$ states, duration of each temporal filter is $d = 60$ seconds, and the output is $C = 7$ classes. Each input image is $108 \times 108 \times 3$.

When using multiple data sources, like phase labels and tool information, auxiliary signals are concatenated per time step with the intermediate spatial features h_t such that we jointly learn temporal filters across all input data. These inputs may be of different magnitudes, especially across modalities, so we normalize the features using the standard deviation over the training set.

6.3.2 Surgical Phase Classifier

Our goal is to predict the best phase labeling $\hat{y} = \{\hat{y}_t\}_{t=1}^T$ given spatiotemporal activations $s = \{s_t\}_{t=1}^T$. We compare several classifiers: a frame-wise linear model, our segmental model as used in Chapter 4, and the Dynamic Time Warping baseline from Chapter 3

1) Linear Model (LM): The output of our ST-CNN, $\hat{Y}_t$, is a set of probabilities corresponding to which phase is active at that time. This model simply takes the most likely phase given the current window of data: $\hat{y}_t = \arg\max_c \hat{Y}_t^c$.

2) Semi-Markov Model (SMM): We jointly infer the start time, end time, and phase

label for each of the M segments in a sequence using a constrained segmental model. Let K be an upper bound on the number of possible segments in a sequence (e.g. 9 for EndoVis) as defined earlier.

3) Time-invariant Model (DTW): DTW captures both the local changes in spatiotemporal activations within each segment as well higher-level temporal ordering of phases. Here we use superscript (i) to indicate each trial. For test sequence $s^{(i)}$ we compute the DTW distance [120] to all training sequences $s^{(j)}$:

$$DTW(s^{(i)}, s^{(j)}) = \min_{c} \sum_{t=1}^{T} \|s_t^{(i)} - s_{c_t}^{(j)}\|_1 \tag{6.4}$$

where $c = \{c_t\}_{t=1}^{T}$ are the correspondences between activations in each sequence. Prediction $\hat{y}^{(i)}$ is computed by propagating the labels from sequence j that has the smallest DTW distance such that $\hat{y}_t^{(i)} = y_{c_t}^{(j)}$ for all times t. This approach was inspired by Padoy *et al.* [115] which achieved high performance on surgical phase recognition from tool usage data on other datasets.

6.4 Datasets

EndoVis Dataset: The EndoVis surgical phase recognition dataset [165], from the Technical University of Munich (TUM), consists of video, tool usage, and surgical phase labels for seven laparoscopic cholecystectomy procedures. The procedures were

performed by a small set of surgeons at the same hospital and have similar workflow. Figure 6.1 shows an example of each phase. In six of seven procedures the phase order is: *Place Trocars*, *Prep*, *Clip/Cut*, *Dissect*, *Retrieval*, *Hemostasis*, *Retrieval*, *Hemostasis*, *Drainage/finish*. In one of the videos, there is only one instance each of *Retrieval* and *Hemostasis*. This dataset also contains tool usage data that was collected by manually labeling whether each instrument was in use at any given time. Tools include: *liver retractor*, *fan retractor*, *alligator forceps*, *PE forceps*, *irrigation rod*, *suction rod*, *scissors*, *retrieval bag*, *plastic clips applicator*, *metal clips applicator*. We evaluate on EndoVis using Leave One Video Out cross validation.

M2CAI 2016 Dataset: There are 27 laparoscopic cholecystectomy procedure videos in the training set and 15 in the test set which were each collected from two university hospitals. We do not have access to the labels in the ground truth set, so we show results computed by performing cross validation on the training set. We created 5 splits, each with about 22 training videos and 5 test videos. Videos were chosen randomly for each split, with the constraint that the same video may only appear in one split. The labels are slightly different than on Endovis. These are *Trocar Placement*,*Preparation*,*Calot Triangle Dissection*,*Clipping Cutting*,*Gallbladder Dissection*,*Gallbladder Packaging*,*Cleaning Coagulation*, and *Gallbladder Retraction*.

EndoTube Dataset: We introduce a new dataset, EndoTube, to address the ability of our models to generalize to real-world surgical environments. We curated videos from Youtube that contain full cholecystectomy procedures and labeled them using

the same surgical phases as EndoVis. All videos include each phase from *Preparing Calots Triangle* through *Gall Bladder Retrieval*, but may not include *insert tools* or *finish*. This dataset contains 25 procedures which were performed at 19 hospitals in 9 countries. Some videos are as short as 4 minutes and jump in time between each of the major phases, while others last up to 27 minutes and show the whole surgery. The average video length is 11.4 minutes. We sifted through dozens of videos and selected ones in which none of the core phases are skipped and the edits did not substantially detract from the video. Some videos are intended for surgical training and have extraneous segments such as powerpoint slides at the beginning. We label these portions as *null* and remove these frames after prediction but before computing accuracy metrics.

Data was manually labeled using the phase definitions from EndoVis by one engineer experienced in the surgical domain. The labels were verified by a second engineer who was very familiar with the EndoVis dataset. We perform 5-fold cross-validation such that we train on 20 instances and test on 5.

Metrics: We evaluate using accuracy and segmental boundary distance. On the M2CAI 2016 dataset we include Jaccard overlap scores, which were used for the competition.

Twinanda *et al.* [2] proposed the boundary distance metric which measures the percentage of the temporal boundaries that are correctly predicted within a certain interval. The motivation is that temporal phase boundaries are often ambiguous and

thus the precise start or end time is not of critical importance. Practically, for each segment, we compute the distance of each true starting time and the closest predicted starting time, and determine if their difference is within a specified threshold. We show results for distance thresholds of $\tau = \{30, 60, 90, 120\}$.

6.5 Results and Discussion

In our first set of experiments, shown in Table 6.1, we compare results using various spatial CNN models on the EndoVis dataset. We include our spatial CNN and spatiotemporal CNN as well as the spatial models used by Twinwanda *et al.* [2]. They (1) fine-tuned an ImageNet-pretrained AlexNet model on EndoVis, (2) fine-tuned that pre-trained AlexNet on a large private dataset of 80 surgical videos (labeled PhaseNet), and (3) augmented the architecture to incorporate tool label information (labeled EndoNet). While our spatial CNN does not outperform the models of theirs which were trained on outside datasets, the ST-CNN does. The jump in performance of our spatial and spatiotemporal CNNs is especially noteworthy. Training a ST-CNN using their outside dataset may improve results even further. We also performed preliminary experiments fine-tuning a VGG network trained on ImageNet. Results were about 7% worse than our spatial CNN, it took longer to compute the activations for each image, and the VGG model required much more memory. Note that our network contains fewer layers and many less parameters so it is much faster to train and test compared to all of these other models.

Model	AlexNet [2]	PhaseNet [2]	EndoNet [2]	Spatial CNN	ST-CNN
Accuracy	56.9	62.6	65.9	57.6	**69.0**

Table 6.1: Performance of different CNN architectures on EndoVis.

EndoVis

Data source(s)	LM	SMM	DTW	[166]	[2]
Video	69.0	77.8	**84.6**	68.1	79.7*
Tools	56.4	78.3	**91.2**	78.9	73.0
Video + Tools	73.3	80.4	**91.0**	88.9	-

EndoTube

Data source	LM	SMM	DTW
Video	56.3	60.1	62.4

Table 6.2: Results from (top) EndoVis and (bottom) EndoTube. *Note: [2] achieves 86.0% on EndoVis when their CNN is pre-trained on a larger surgical dataset and with tool information.

Table 6.2 shows our accuracy results on EndoVis and EndoTube datasets. Each row was trained using either video, tool information, or both. Recall, when using video, the auxiliary term Z in the spatial component is the set of phase labels at each time step. For the multi-modal results we concatenated the video and tool features before feeding them into the temporal CNN component.

We achieve state-of-the-art results when only using tool data, when combining tool and video data, and when only using video (assuming no pre-training). Our high tool-only results are consistent with the findings of Padoy *et al.* [115] on another Cholecystectomy dataset. Twinanda *et al.* [2] perform better than our results when they train on an unpublished surgical dataset, however, when training on EndoVis our video-based results are better. We see that the Spatiotemporal CNN performs favorably compared to a spatial CNN or using the hand-crafted features by Dergachy-

ova *et al.* [166]. Twinanda *et al.* [2] achieve 56.9% accuracy using AlexNet, 62.6% using a spatial CNN trained on EndoVis, and 65.9% when training on both image and tools. For comparison, our ST-CNN, without a high level temporal model (LM), achieves 69.01%. Furthermore, we see that the DTW-based model achieves notably higher accuracy than the linear or semi-Markov models. DTW captures how the spatiotemporal activations change within each phase which appears to have a large impact on accuracy.

Performance on EndoTube (62.4%) is far lower than EndoVis, but is commensurate with the large increase of variability. We analyzed the results from individual videos and found, on average, the best sequence in each split achieves 90.7% accuracy and worst sequence achieves 33.8%. We achieve worst performance when the video quality is low (e.g. abnormally high contrast) and when the surgical tools look substantially different than normal. Three of the worst sequences have atypical clipping phases. One surgeon uses thread instead of clips, another uses a unique style of clips, and the third does not use any clips. Despite poor accuracy, we think it is important to include these videos because they address real-world concerns with large-scale workflow analysis.

Table 6.3 shows the percentage of EndoVis phases that are within τ seconds from the true phase starting times. We see that most of the time the predictions are correct within a reasonable tolerance. These videos are on average 41 minutes, so if a prediction is correct within 120 seconds, the phase shift accounts for less than 5% of

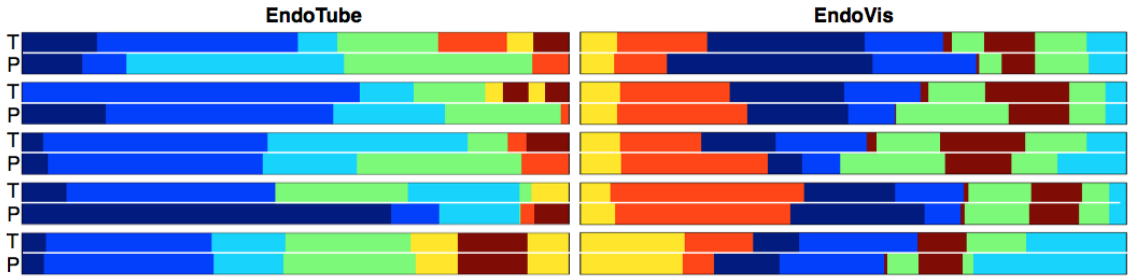


Figure 6.4: Example predictions from the EndoTube and EndoVis datasets. The top of each plot depicts the sequence of true phases and the bottom depicts the predicted labels using Dynamic Time Warping. Each color corresponds to a unique surgical phase.

the video. When combining video and tool-use, all boundaries are correctly predicted within 3 minutes.

Features	≤ 30	≤ 60	≤ 90	≤ 120	≤ 150	≤ 180
Video	66.2	76.1	82.5	88.8	93.6	93.6
Tools	**90.4**	**90.4**	92.0	93.6	93.6	95.2
Video+Tools	85.2	**90.4**	**92.0**	**95.2**	**98.4**	**100.0**

Table 6.3: The percentage of predicted label boundaries within the specified distance (in seconds) to the true boundaries on EndoVis using the DTW model.

Figure 6.4 shows predictions from EndoVis and EndoTube. Qualitatively, we see that many errors in accuracy can be attributed to small temporal shifts. On EndoTube, some predictions (e.g. rows 3 & 5) perform very well whereas others (e.g. rows 1 & 4) perform poorly.

6.5.1 M2CAI 2016

For M2CAI 2016, we evaluated on the new competition dataset and provided more comparative results on EndoVis. We added results using the Latent Convolutional

M2CAI 2016 Dataset

Phases	TP	Prep	CTD	Clip	GD	GP	CC	GR	Average
LM	62.7	18.7	46.2	26.9	40.6	13.5	20.4	34.3	32.9±16.4
DTW	65.6	27.3	56.0	35.2	46.1	13.5	21.9	35.8	37.7±17.4
LC-SC-CRF	70.3	21.5	60.0	34.7	**59.1**	16.6	39.0	44.2	43.2±19.0
ST-CNN	74.8	26.1	56.1	36.8	52.8	**31.2**	37.1	53.4	46.0±16.0
ED-TCN	**83.0**	**54.7**	**75.3**	**50.6**	51.6	30.1	**56.7**	**70.9**	**59.1**±16.7

Table 6.4: Jaccard scores for each phase in the M2CAI dataset. Phases: TP=Trocar Placement, Prep=Preparation, CTD=Calot Triangle Dissection,Clip=Clipping Cutting, GD=Gallbladder Dissection, GP=Gallbladder Packaging,CC=Cleaning Coagulation, GR=Gallbladder Retraction

Skip Chain CRF (LC-SC-CRF) from Chapter 3 and our Encoder-Decoder Temporal Convolutional Network (ED-TCN). The results shown here were all performed in offline (acausal) mode. Note there are a couple of discrepancies with the results from before. Namely, the ST-CNN results did not use segmental inference and DTW was normalized differently. Previously, for the DTW method, we resampled all sequences to be of the same canonical length – which actually achieved slightly better performance – whereas in these experiments we do not resample.

Table 6.4 shows the Jaccard scores for each phase in the M2CAI dataset, Table 6.5 shows Jaccard scores for EndoVis using the spatial video features and using the sensor data. Table 6.6 shows the average accuracies for M2CAI along with the standard deviation for each.

On the M2CAI dataset the ED-TCN outperforms all other models. While both the CRF and ST-CNN encode the relationship between the current action and the inferred previous action, we find that the hierarchy of convolutional filters in the TCN is better able to capture temporal information. This is consistent with our findings

EndoVis dataset: Video-only Jaccard Scores

Model	Clip	DG	DC	Hemo	PT	Prep	RG	Average
LM	35.2	37.1	34.7	32.5	26.2	43.9	41.4	35.9 ± 05.8
DTW	**59.6**	**63.7**	**88.9**	**66.1**	**82.5**	**77.5**	**79.8**	**74.0** ± 10.9
LC-SC-CRF	53.5	49.9	76.9	45.6	73.0	73.5	70.2	63.2 ± 13.0
ST-CNN	53.6	50.3	78.6	38.4	60.4	74.2	71.9	61.1 ± 14.6
ED-TCN	51.1	50.2	79.6	46.5	72.8	71.9	67.1	62.8 ± 13.2

EndoVis dataset: Sensor-only Jaccard Scores

Model	Clip	DG	DC	Hemo	PT	Prep	RG	Average
LM	56.2	34.6	00.0	76.4	47.7	02.4	67.4	40.7 ± 30.1
DTW	56.4	**68.9**	86.9	72.2	**99.3**	**88.2**	**83.9**	79.4 ± 14.3
LC-SC-CRF	40.1	69.1	80.3	80.7	88.9	64.4	76.9	71.5 ± 16.0
ST-CNN	68.1	42.4	79.7	83.1	78.4	33.3	70.8	65.1 ± 19.5
ED-TCN	**68.8**	64.8	**91.9**	**90.4**	95.5	81.2	77.3	**81.4** ± 11.8

Table 6.5: Jaccard scores for each phase in the EndoVis dataset. Phases: CC=Clipping Cutting, DG=Dissection Gallbladder, DC=Drainage And Closing, Hemo=Hemostasis, PT=Placement Trocars, Prep=Preparation,RG=Retrieving Gallbladder

Frame-wise Accuracy Results

Model	M2CAI	EndoVis Video	EndoVis Sensors
LM	55.8±10.1	56.0±6.0	59.4±12.4
DTW	55.3±17.9	**83.1**±3.9	85.4± 9.2
LC-SC-CRF	69.4±09.7	76.5±7.0	79.2± 9.4
ST-CNN	67.1±14.3	75.8±9.3	72.3±12.9
ED-TCN (acausal)	**74.5**±11.9	76.0±7.0	**85.7**±13.8

Table 6.6: Averaged frame-wise accuracy on the video-based M2CAI dataset and using video or sensor data on EndoVis.

in Chapter 5.

It is important to note that results vary significantly using slightly different hyper parameters. For example, accuracy on the sensor-based DTW results vary between mid-80s and low 90s using different sampling schemes. If we resample all time-steps so that sequences are of the same canonical length, as done in the earlier set of results, we achieve higher sensor results but lower video results. Even sampling every 10 frames versus every 20 frames has a noticeable, albeit smaller, impact. This makes us hesitant to trust the usefulness of results on this dataset.

Figure 6.5 compares the ground truth phases with our predictions for two of our cross validation splits on the M2CAI data. Notice that many of our errors come from small temporal offsets. For example, the bottom-most example in each set looks very good, but each gets less than 90% accuracy.

While our methods outperform the baselines, there is still significant work that needs to be done to achieve the level of performance necessary for using this in hospital settings. In the near-term, we plan to focus on better understanding where are errors are coming from. For example, we would like to better understand why, on occasion, the classifier detects phases out of order. It is unclear if this is due to an abnormality in the video, for example an organ whose appearance varies substantially from the norm, or if it is in a limitation of our model. Identifying these error modes will be crucial for furthering this line of work.

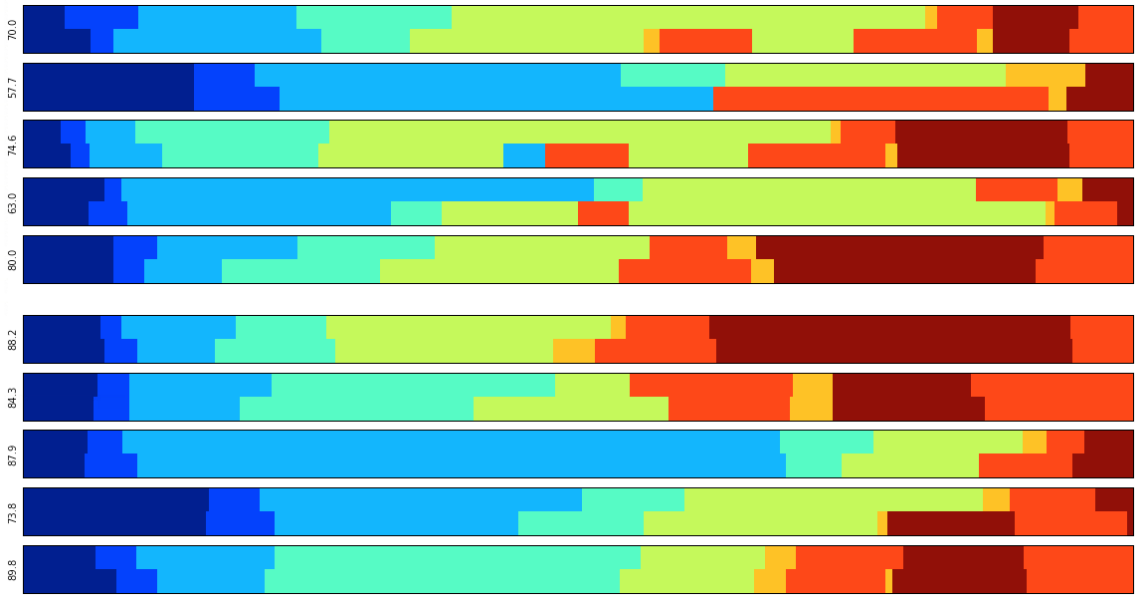


Figure 6.5: Results from two of the M2CAI cross validation splits. The top of each plot is the sequence of ground truth phases and the bottom is our predicted phases using TCN. The x-axis indicates time, which is normalized for visualization purposes. The text on the left indicates the frame-wise accuracy for each split. Each of the M2CAI phases corresponds to one color, from dark blue to dark red, in the order *Trocar Placement*, *Preparation*, *Calot Triangle Dissection*, *Clipping Cutting*, *Gallbladder Dissection*, *Gallbladder Packaging*, *Cleaning Coagulation*, and *Gallbladder Retraction*.

6.6 Conclusion

In summary, we make three important observations about surgical workflow analysis. First, despite high performance on single-institution datasets like EndoVis, current models are insufficient for handling the variability on multi-institution datasets like EndoTube. This is a result of an insufficient quantity of data and limitations with the model. Perhaps, new data augmentation techniques could improve performance on these videos. Second, explicitly capturing local temporal information, such as with Spatiotemporal CNN, can improve performance compared to traditional spatial CNNs. Lastly, on small datasets like EndoVis, simple DTW-based models, which jointly capture how our spatiotemporal activations change across time both locally and globally, are beneficial. However, when there is more data or variation between videos, as in the M2CAI 2016 dataset, then our TCNs achieve best results.

Chapter 7

Conclusions and Future Work

The overarching goal of this thesis was to work towards bridging the gap in action segmentation performance between approaches that rely on domain-specific sensing to solutions that operate solely using video. Through the introduction of temporal and spatiotemporal models we were able to improve sensor- and video-based performance across several datasets and metrics.

In Chapter 3 we introduced a strong baseline model for sensor-based performance, the Latent Convolutional Skip Chain CRF, which improved upon the state of the art in two important ways. First, we showed that skip-chains are a simple, efficient and effective mechanism for capturing the pairwise temporal relationships between actions. Skip chains compensate for the disconnect between the action rate – how frequently actions transition – and the frame rate – how frequently the data is sampled. We

showed empirically that this model outperforms more complicated segmental models and described why it does so. Second, we introduced the notion of a convolutional action primitive which captures how the input sensor signal changes as a function of the given action. The learned primitives are capable of capturing complex patterns, as we visualized, and were relatively interpretable. We showed that these action primitives can be learned jointly with the skip chain parameters using a Latent Structural SVM.

In Chapter 4 we extended these convolutional filters to video with the introduction of a spatiotemporal CNN, which factorizes the input into a component that captures the spatial relationships between objects and a component that captures how these relationships change over time. This approach outperformed common video baselines from the large-scale action classification literature on our tasks by a wide margin. In addition, we showed that while pre-trained CNNs work well for many other vision applications, they are not very effective in situated tasks like ours. We introduced an approach called sensor substitution for training spatial CNNs solely using sensor data. This approach can be used for pre-training CNNs on large unlabeled datasets or for regressing sensor signals directly from video.

One shortcoming of the previous models is that they all consisted of two components: a set of low-level action primitives and a high-level classifier that captured long-range temporal patterns. In Chapter 5 we introduced a hierarchical variation of our convolutional model capable of capturing both low- and high-level patterns. We

also showed that can capture complex dependencies such as action compositions and time-delays. We were able to obtain video-based action segmentation results that were comparable to those operating on domain-specific sensors using TCNs with our spatial CNN features. This has important implications for tasks like surgical skill assessment, because our models can now be used with non-robotic surgical training setups.

Finally, in Chapter 6, we made advances to the area of surgical workflow analysis through the application of the described models and through the creation of a dataset consisting of surgical videos in-the-wild. We showed that despite achieving state-of-the-art performance on constrained datasets, there are still technical hurdles towards applying these models in the real world. While our temporal models appear to capture many of the high-level temporal patterns well, the low-level video features do not model the large variability in equipment and lighting conditions across hospitals.

Aside from the technical innovations, one of the most important contributions comes from our investigation of appropriate metrics for action segmentation. Ultimately we found that the metrics used most commonly in the literature may not correlate with expected real-world performance. In Chapter 3 we described how a model may achieve high accuracy but have many over-segmentation errors. These errors may be detrimental for real-time applications in robotics or when trying to compare the order of actions like in surgical skill evaluation. In this case, we found that Segmental Edit scores, which penalize over-segmentation errors, are an effective

way of evaluating goal-driven activities. In Chapter 6 we went one more step with the introduction of a Segmental F1 score which is broadly applicable to both action segmentation and detection tasks. This metric also penalizes for over-segmentation issues but is more relevant for surveillance-style tasks, where each action is relatively independent and thus the global action ordering is not very important. We hope that other researchers continue to use both of these metrics.

Limitations and Future Work

Despite our advances to the area of action segmentation, there are still limitations which should be addressed by future work.

Level of Supervision: Ultimately, the temporal models we developed required labeled training data for every frame in each dataset. These annotations unfortunately can be very time consuming to obtain. One of the reasons why large datasets like ActivityNet do not have as many action segmentation labels – as opposed to action classification labels – is because they are costly to label.

One solution to this is to look towards recent work on semi-supervised methods in the machine learning literature. The most popular example is Connectionist Temporal Classification (CTC) [171], which is used with RNNs. CTC is a loss function that takes a sparse sequence of labels (e.g. an ordered list of actions), computes the probability that each frame belongs to one of the ordered actions, and outputs a

per-class loss for that sequence. It may be much more economical to label the action ordering than it is to label the action at each frame. This approach is especially relevant for tasks, like surgical skill evaluation, where we are only interested in the order in which actions occurred and not necessarily a per-frame labeling of the actions.

Fixed action vocabulary: Throughout this thesis we assumed a fixed set of action classes – including, in some cases, a background class. While this is a reasonable assumption for goal-driven activities, there are many use cases where this assumption is not valid. For example, let's say we want to take a cooking video and generate a recipe based on the actions performed by the user. There may be actions that we have never seen before – for example, if a person is making creme brulee, they must use a torch to crystallize the sugar on the top of the dessert. If we have never seen the `torch` action then our current approach would incorrectly label the segment corresponding to torch as another class in our vocabulary.

In this setting, it would be advantageous to identify when a new or unseen action is occurring and to label it as such. One way of going about this is to assign a confidence to every segment. While deep networks do typically output a score for every prediction, they are often unreliable. We looked at an example test output using the ED-TCN on 50 Salads and found that on average, when the classifier chose the wrong prediction, the confidence for those wrong predictions was 90.3%.

Single-action: Our models assume that each frame is labeled with only one action class. This assumption is invalid in cases where there are multiple users in a video

and they are performing different actions. If we want to detect multiple actions then a simple way of overcoming this limitation is to replace the softmax function in our ST-CNN or ED-TCN with a sigmoid, which then predicts the probability of every action being on or off, as opposed to only predicting the action that is most likely.

A better approach would be to identify the different users in the scene and then individually predict the action of each user. This, however, requires that you correctly identify the users.

Multi-Modal Representations: Most of our models operated using either video or sensor data. These sensing modalities tend to be complimentary; there are some actions which are much easier to detect from sensors and some that are much easier to detect from video. For example, the video is able to capture a user starting to pick up an action, but the sensors (e.g., accelerometers) will not pick up any motion from that object until the user has touched it. Thus, it may be advantageous to encode the temporal patterns independently and then merge them. One way of combining these using an ED-TCN is to learn one encoder for each modality, concatenate the features, and learn a single decoder.

Interactive Evaluation: The work described in this thesis was developed entirely using well-developed datasets. In some regards this makes it easier to develop new algorithms, however, it often makes it more difficult to fully understand limitations of each model. In other collaborations of mine, including work on collaborative robots for manufacturing, as described in Chapter 1, we were required to perform tasks

in real-time. When evaluating your software – whether a robot platform or action segmentation algorithm – it quickly becomes apparent what the limitations are. It is easier to test for failure modes. In a canned dataset you cannot try out new test cases to find precisely where the algorithm is going wrong.

One solution to this is to define an action segmentation task that can be done easily in the lab, create a small dataset using data collected on this task, train a model, and then perform the task again using a real-time version of our software. During this interactive evaluation it is possible to visualize the predictions and internal representations as you complete the task. For example in a cooking task, using visualizations similar to those shown with the ST-CNN, you can verify that certain tools, like the peeler, are receiving high activations.

Bibliography

[1] C. Lea, R. Vidal, and G. D. Hager, "Learning convolutional action primitives for fine-grained action recognition," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[2] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. de Mathelin, and N. Padoy, "Endonet: A deep architecture for recognition tasks on laparoscopic videos," *CoRR*, vol. abs/1602.03012, 2016. [Online]. Available: http://arxiv.org/abs/1602.03012

[3] A. Fathi, R. Xiaofeng, and J. M. Rehg, "Learning to recognize objects in egocentric activities," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[4] J. Lei, X. Ren, and D. Fox, "Fine-grained kitchen activity recognition using RGB-D," in *ACM Conference on Ubiquitous Computing (Ubicomp)*, 2012, pp. 208–211. [Online]. Available: http://doi.acm.org/10.1145/2370216.2370248

[5] N. N. Vo and A. F. Bobick, "From stochastic grammar to bayes network:

Probabilistic parsing of complex activity," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 2641–2648. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.338

[6] K. P. Hawkins, S. Bansal, N. N. Vo, and A. F. Bobick, "Anticipating human actions for collaboration in the presence of task and sensor uncertainty," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[7] M. Jain, J. C. van Gemert, and C. G. M. Snoek, "What do 15,000 object categories tell us about classifying and localizing actions?" in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[8] F. Caba Heilbron, V. Escorcia, B. Ghanem, and J. Carlos Niebles, "ActivityNet: A large-scale video benchmark for human activity understanding," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[9] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, 2013. [Online]. Available: http://hal.inria.fr/hal-00873267

[10] M. Shokoohi-Yekta, J. Wang, and E. J. Keogh, "On the non-trivial generalization of dynamic time warping to the multi-dimensional case," in *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, 2015, pp. 289–297. [Online]. Available: http://dx.doi.org/10.1137/1.9781611974010.33

[11] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[12] S. Stein and S. J. McKenna, "Combining embedded accelerometers with computer vision for recognizing food preparation activities," in *ACM Conference on Ubiquitous Computing (Ubicomp)*, 2013.

[13] C. Lea, J. Fackler, G. D. Hager, R. Taylor, and S. Saria, "3d sensing algorithms towards building an intelligent intensive care unit," in *AMIA Clinical Research Informatics*, 2013.

[14] C. Lea, J. Fackler, G. D. Hager, and R. Taylor, "Towards automated activity recognition in an intensive care unit," in *MICCAI Workshop on Modeling and Monitoring of Computer Assisted Interventions (M2CAI)*, 2013.

[15] C. Lea, G. D. Hager, and R. Vidal, "An improved model for segmentation and recognition of fine-grained activities with application to surgical training tasks," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2015.

[16] R. DiPietro, C. Lea, A. Malpani, N. Ahmidi, S. S. Vedula, G. I. Lee, M. R. Lee, and G. D. Hager, "Recognizing surgical activities with recurrent neural networks," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2016.

[17] C. B. Barden, M. C. Specht, M. D. McCarter, J. M. Daly, and T. J. Fahey, "Effects of limited work hours on surgical training," *J. Am. Coll. Surg.*, vol. 195, no. 4, pp. 531–538, Oct 2002.

[18] K. Cleary, A. Kinsella, and S. K. Mun, "OR 2020 workshop report: Operating room of the future," 2005, CARS: Computer Assisted Radiology and Surgery.

[19] Y. Gao, S. S. Vedula, C. E. Reiley, N. Ahmidi, B. Varadarajan, H. C. Lin, L. Tao, L. Zappella, B. Béjar, D. D. Yuh *et al.*, "JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS): A surgical activity dataset for human motion modeling," in *MICCAI Workshop: M2CAI*, 2014.

[20] C. Lea, J. H. Choi, A. Reiter, and G. Hager, "Surgical phase recognition: from instrumented ors to hospitals around the world," in *MICCAI Workshop on Modeling and Monitoring of Computer Assisted Interventions (M2CAI)*, 2016.

[21] A. Malpani, C. Lea, C. C. G. Chen, and G. D. Hager, "System events: Readily accessible features for surgical phase detection," in *International Conference on Information Processing in Computer-Assisted Interventions (IPCAI)*, 2016.

[22] K. Ramirez-Amaro, M. Beetz, and G. Cheng, "Automatic segmentation and recognition of human activities from observation based on semantic reasoning," in *IEEE Conference on Intelligent Robots and Systems (IROS)*, 2014.

[23] A. Fathi, A. Farhadi, and J. M. Rehg, "Understanding egocentric activities,"

in *IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 407–414. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2011.6126269

[24] B. Singh, T. K. Marks, M. Jones, O. Tuzel, and M. Shao, "A multi-stream bi-directional recurrent neural network for fine-grained action detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] I. Dianov, K. Ramirez-Amaro, and G. Cheng, "Generating compact models for traffic scenarios to estimate driver behavior using semantic reasoning," 2016.

[26] K. Ramirez-Amaro, E.-S. Kim, J. Kim, B.-T. Zhang, M. Beetz, and G. Cheng, "Enhancing Human Action Recognition through Spatio-temporal Feature Learning and Semantic Rules," in *IEEE-RAS International Conference on Humanoid Robots*, October 2013.

[27] N. H. Kirk, K. Ramírez-Amaro, E. Dean-León, M. Saveriano, and G. Cheng, "Online prediction of activities with structure: Exploiting contextual associations and sequences," in *Humanoid Robots (Humanoids), IEEE-RAS 15th International Conference on.* IEEE, 2015, pp. 744–749.

[28] K. Ramirez-Amaro, M. Beetz, and G. Cheng, "Understanding the intention of human activities through semantic perception: observation, understanding and execution on a humanoid robot," *Advanced Robotics*, vol. 29, no. 5, pp. 345–362, 2015.

[29] S. Stein and S. J. McKenna, "User-adaptive models for recognizing food preparation activities," in *ACM International Conference on Multimedia, Workshop on Multimedia for Cooking and Eating Activities (CEA)*. ACM, 2013.

[30] L. Tao, E. Elhamifar, S. Khudanpur, G. D. Hager, and R. Vidal, "Sparse hidden markov models for surgical gesture classification and skill evaluation." in *International Conference on Information Processing in Computer-Assisted Interventions (IPCAI)*, 2012.

[31] B. Varadarajan, "Learning and inference algorithms for dynamical system models of dextrous motion," Ph.D. dissertation, Johns Hopkins University, 2011.

[32] J. Rosen, J. Brown, L. Chang, M. Sinanan, and B. Hannaford, "Generalized approach for modeling minimally invasive surgery as a stochastic process using a discrete markov model," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 3, pp. 399–413, March 2006.

[33] L. Tao, L. Zappella, G. D. Hager, and R. Vidal, "Surgical gesture segmentation and recognition." in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2013, pp. 339–346.

[34] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. G. G. D. Hager, "Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning," in *International Symposium on Robotic Research (ISRR)*, 2015.

[35] A. Zia, Y. Sharma, V. Bettadapura, E. Sarin, M. Clements, and I. Essa, "Automated assessment of surgical skills using frequency analysis," in *International Conference on Medical Image Computing and Computer Assisted Interventions (MICCAI)*, October 2015.

[36] H. S. Koppula and A. Saxena, "Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation," in *International Conference on Machine Learning (ICML)*, 2013.

[37] H. Koppula and A. Saxena, "Anticipating human activities using object affordances for reactive robotic response," in *IEEE Trans PAMI*, 2015.

[38] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[39] A. Jain, A. Singh, H. S. Koppula, S. Soh, and A. Saxena, "Recurrent neural networks for driver activity anticipation via sensory-fusion architecture," in *International Conference on Robotics and Automation (ICRA)*, 2016.

[40] F. De la Torre Frade, J. K. Hodgins , A. W. Bargteil, X. Martin Artal, J. C. Macey, A. Collado I Castells, and J. Beltran, "Guide to the carnegie mellon university multimodal activity (cmu-mmac) database," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-22, April 2008.

[41] E. H. Spriggs, F. D. L. Torre, and M. Hebert, "Temporal segmentation and activity classification from first-person sensing," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2009.

[42] J. Carvajal, C. McCool, B. C. Lovell, and C. Sanderson, "Joint recognition and segmentation of actions via probabilistic integration of spatio-temporal fisher vectors," *CoRR*, vol. abs/1602.01601, 2016. [Online]. Available: http://arxiv.org/abs/1602.01601

[43] Y. Yang, C. Fermuller, and Y. Aloimonos, "Detection of manipulation action consequences (mac)," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 2563–2570. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2013.331

[44] A. Guha, Y. Yang, C. Fermuuller, and Y. Aloimonos, "Minimalist plans for interpreting manipulation actions," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 2013, pp. 5908–5914.

[45] Y. Yang, Y. Aloimonos, C. Fermller, and E. E. Aksoy, "Learning the semantics of manipulation action." in *ACL*. The Association for Computer Linguistics, 2015, pp. 676–686.

[46] Y. Yang, C. Fermller, Y. Li, and Y. Aloimonos, "Grasp type revisited: A

modern perspective on a classical feature for vision," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[47] K. Zampogiannis, Y. Yang, C. Fermller, and Y. Aloimonos, "Learning the spatial semantics of manipulation actions through preposition grounding." in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[48] L. Sun, K. Jia, D.-Y. Yeung, and B. Shi, "Human action recognition using factorized spatio-temporal convolutional networks," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[49] M. Rohrbach, A. Rohrbach, M. Regneri, S. Amin, M. Andriluka, M. Pinkal, and B. Schiele, "Recognizing fine-grained and composite activities using hand-centric features and script data," *International Journal of Computer Vision (IJCV)*, 2015.

[50] A. Fathi and J. M. Rehg, "Modeling actions through state changes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 2579–2586. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2013.333

[51] Y. Li, Z. Ye, and J. M. Rehg, "Delving into egocentric actions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[52] G. Cheron, I. Laptev, and C. Schmid, "P-cnn: Pose-based cnn features for

action recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[53] B. Ni, V. R. Paramathayalan, and P. Moulin, "Multiple granularity analysis for fine-grained action detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[54] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local svm approach," in *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ser. ICPR '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 32–36. [Online]. Available: http://dx.doi.org/10.1109/ICPR.2004.747

[55] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 12, pp. 2247–2253, December 2007.

[56] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Action Recognition by Dense Trajectories," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, United States, Jun. 2011, pp. 3169–3176. [Online]. Available: http://hal.inria.fr/inria-00583818/en

[57] H. Pirsiavash and D. Ramanan, "Parsing videos of actions with segmental grammars," in *IEEE Conference on Computer Vision and Pattern*

Recognition (CVPR), 2014, pp. 612–619. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.85

[58] I. Laptev, "On space-time interest points," *International Journal of Computer Vision (IJCV)*, vol. 64, no. 2-3, pp. 107–123, 2005. [Online]. Available: http://dx.doi.org/10.1007/s11263-005-1838-7

[59] A. Kläser, M. Marszałek, and C. Schmid, "A spatio-temporal descriptor based on 3d-gradients," in *British Machine Vision Conference (BMVC)*, sep 2008, pp. 995–1004. [Online]. Available: http://lear.inrialpes.fr/pubs/2008/KMS08

[60] L. Pishchulin, M. Andriluka, and B. Schiele, "Fine-grained activity recognition with holistic and pose based features," in *German Conference on Pattern Recognition (GCPR)*, 2014.

[61] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[62] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems (NIPS)*, 2014. [Online]. Available: http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos

[63] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spa-

tiotemporal features with 3d convolutional networks," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[64] X. Peng and C. Schmid, "Encoding feature maps of cnns for action recognition," in *CVPR, THUMOS Challenge 2015 Workshop*, 2015.

[65] J. Y. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[66] C. Sutton and A. Mccallum, *Introduction to Conditional Random Fields for Relational Learning.* MIT Press, 2006.

[67] S. Nowozin, C. Rother, S. Bagon, T. Sharp, and P. Kohli, "Decision tree fields," *2011 International Conference on Computer Vision*, pp. 1668–1675, Nov. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6126429

[68] N. Hu, G. Englebienne, Z. Lou, and B. Kröse, "Learning latent structure for activity recognition," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[69] K. Tang, L. Fei-Fei, and D. Koller, "Learning latent temporal structure for complex event detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[70] J. D. Ferguson, "Variable duration models for speech," *In Proc. Symposium on the Application of Hidden Markov Models to Text and Speech*, p. 143179, 1980.

[71] S. Sarawagi and W. W. Cohen, "Semi-markov conditional random fields for information extraction," in *Advances in Neural Information Processing Systems (NIPS)*, 2004. [Online]. Available: http://books.nips.cc/papers/files/nips17/NIPS2004_0427.pdf

[72] S.-Z. Yu, "Hidden semi-markov models," *Artificial Intelligence*, vol. 174, no. 2, pp. 215 – 243, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370209001416

[73] T. van Kasteren, G. Englebienne, and B. J. Kröse, "Activity recognition using semi-markov models on real world smart home datasets," *Journal of Ambient Intelligence and Smart Environments*, vol. 2, no. 3, pp. 311–325, 2010.

[74] S. Lee, H. X. Le, H. Q. Ngo, H. I. Kim, M. Han, Y.-K. Lee *et al.*, "Semi-markov conditional random fields for accelerometer-based activity recognition," *Applied Intelligence*, vol. 35, no. 2, pp. 226–241, 2011.

[75] Q. Shi, L. Cheng, L. Wang, and A. Smola, "Human action segmentation and recognition using discriminative semi-markov models," *International Journal of Computer Vision (IJCV)*, vol. 93, no. 1, pp. 22–32, 2011.

[76] J. Ferguson, "Hidden markov models for speech," *IDA, Princeton, NJ*, 1980.

[77] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *PROCEEDINGS OF THE IEEE*, 1989, pp. 257–286.

[78] B.-H. Juang and L. Rabiner, "Mixture autoregressive hidden markov models for speech signals," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 33, no. 6, pp. 1404–1413, 1985.

[79] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*. Springer Publishing Company, Incorporated, 2014.

[80] K. Murphy, "Dynamic Bayesian Networks: Representation, Inference and Learning," Ph.D. dissertation, UC Berkeley, Computer Science Division, 2002.

[81] R. Vidal, A. Chiuso, and S. Soatto, "Application of hybrid system identification in computer vision," in *Proceedings of the European Control Conference*, 2007, pp. 27–34.

[82] A. Bissacco, A. Chiuso, Y. Ma, and S. Soatto, "Recognition of human gaits," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, December 2001, pp. 52–57.

[83] R. Chaudhry, A. Ravich, G. Hager, and R. Vidal, "Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions," in *in In IEEE Conference on Computer Vision and Pattern Recognition (CVPR*, 2009.

[84] S. V. N. Vishwanathan, A. J. Smola, and R. Vidal, "Binet-cauchy kernels on dynamical systems and its application to the analysis of dynamic scenes," *International Journal of Computer Vision*, vol. 73, no. 1, pp. 95–119, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11263-006-9352-0

[85] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, A. Y. Hannun, B. Jun, T. Han, P. LeGresley, X. Li, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, S. Qian, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, C. Wang, Y. Wang, Z. Wang, B. Xiao, Y. Xie, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2 : End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning (ICML)*, 2016. [Online]. Available: http://jmlr.org/proceedings/papers/v48/amodei16.html

[86] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[87] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[88] A. Y. Hannun, C. Case, J. Casper, B. C. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014. [Online]. Available: http://arxiv.org/abs/1412.5567

[89] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition," in *INTERSPEECH International Speech Communication Association*, 2013. [Online]. Available: http://www.isca-speech.org/archive/interspeech_2013/i13_3366.html

[90] J. Zhang, W. Li, P. O. Ogunbona, P. Wang, and C. Tang, "Rgb-d-based action recognition datasets: A survey," *CoRR*, vol. abs/1601.05511, 2016. [Online]. Available: http://arxiv.org/abs/1601.05511

[91] Y. Gao, S. S. Vedula, C. E. Reiley, N. Ahmidi, B. Varadarajan, H. C. Lin, L. Tao, L. Zappella, B. Bejar, D. D. Yuh, C. C. G. Chen, R. Vidal, S. Khudanpur, and G. D. Hager, "The jhu-isi gesture and skill assessment dataset (jigsaws): A surgical activity working set for human motion modeling," in *Medical Image Computing and Computer-Assisted Intervention M2CAI - MICCAI Workshop*, 2014.

[92] S. Singh, C. Arora, and C. V. Jawahar, "First person action recognition using

deep learned descriptors," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[93] E. H. S. Taralova, F. De la Torre, and M. Hebert, "Temporal segmentation and activity classification from first-person sensing," in *IEEE Workshop on Egocentric Vision, in conjunction with CVPR 2009*, June 2009.

[94] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., 2012.

[95] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference Learning Representations (ICLR)*, 2015.

[96] C.-N. Yu and T. Joachims, "Learning structural svms with latent variables," in *International Conference on Machine Learning (ICML)*, 2009.

[97] H. Kuehne, J. Gall, and T. Serre, "An end-to-end generative framework for video segmentation and recognition," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, Lake Placid, Mar 2016.

[98] N. V. Cuong, N. Ye, W. S. Lee, and H. L. Chieu, "Conditional random field with high-order dependencies for sequence labeling and segmentation," *J.*

Mach. Learn. Res., vol. 15, no. 1, pp. 981–1009, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2638567

[99] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in Computational Intelligence. Springer, 2012, vol. 385. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24797-2

[100] J. Koutník, K. Greff, F. J. Gomez, and Jürgen Schmidhuber, "A clockwork rnn," in *International Conference on Machine Learning (ICML)*, 2014.

[101] L. Kong, C. Dyer, and N. A. Smith, "Segmental recurrent neural networks," *CoRR*, vol. abs/1511.06018, 2015. [Online]. Available: http://arxiv.org/abs/1511.06018

[102] C. Sutton and A. McCallum, "Collective segmentation and labeling of distant entities in information extraction," 2004, presented at ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields. Banff, Canada. Also Umass CS TR 04-49.

[103] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[104] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large Margin Methods for Structured and Interdependent Output Variables," *Journal*

of Machine Learning Research (JMLR), vol. 6, pp. 1453–1484, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.6373&rep=rep1&type=pdf

[105] Y. Wang and G. Mori, "Hidden part models for human action recognition: Probabilistic vs. max-margin," 2011.

[106] A. Müller and S. Behnke, "Learning a Loopy Model For Semantic Segmentation Exactly," *arXiv preprint arXiv:1309.4061*, no. 2006, 2013. [Online]. Available: http://arxiv.org/abs/1309.4061

[107] S. Nowozin, "Structured Learning and Prediction in Computer Vision," *Foundations and Trends in Computer Graphics and Vision*, vol. 6, no. 3-4, pp. 185–365, 2010. [Online]. Available: http://www.nowpublishers.com/product.aspx?product=CGV&doi=0600000033

[108] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-24, 2010.

[109] H. Lobel, R. Vidal, and A. Soto, "Learning shared, discriminative, and compact representations for visual recognition," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 2015.

[110] Y. B. Ian Goodfellow and A. Courville, "Deep learning," 2016, book in

preparation for MIT Press. [Online]. Available: http://goodfeli.github.io/dlbook/

[111] M. K. Titsias, C. C. Holmes, and C. Yau, "Statistical Inference in Hidden Markov Models Using k-Segment Constraints," *Journal of the American Statistical Association*, vol. 111, no. 513, pp. 200–215, 2016, pMID: 27226674. [Online]. Available: http://dx.doi.org/10.1080/01621459.2014.998762

[112] C. Lea, A. Reiter, R. Vidal, and G. D. Hager, "Segmental spatio-temporal CNNs for fine-grained action segmentation," *European Conference on Computer Vision (ECCV)*, 2016.

[113] G. Andrew, "A hybrid markov/semi-markov conditional random field for sequence segmentation," in *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, January 2006.

[114] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[115] N. Padoy, T. Blum, S.-A. Ahmadi, H. Feussner, M.-O. Berger, and N. Navab, "Statistical modeling and recognition of surgical workflow," *Medical Image Analysis (MedIA)*, 2012, computer Assisted Interventions.

[116] F. Zhou and F. De la Torre, "Canonical time warping for alignment of human behavior," in *Advances in Neural Information Processing Systems Conference (NIPS)*, December 2009.

[117] ——, "Generalized time warping for multi-modal alignment of human motion," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[118] M. Hoai and F. De la Torre, "Maximum margin temporal clustering," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[119] K. Kulkarni, G. Evangelidis, J. Cech, and R. Horaud, "Continuous action recognition based on sequence alignment," *International Journal of Computer Vision*, vol. 112, no. 1, pp. 90–114, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11263-014-0758-9

[120] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb 1978.

[121] E. Keogh and A. C. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, vol. 7, no. 3, pp. 358–386, 2005. [Online]. Available: http://dx.doi.org/10.1007/s10115-004-0154-9

[122] D. F. Silva and G. E. A. P. A. Batista, *Speeding Up All-Pairwise Dynamic*

Time Warping Matrix Calculation, ch. 94, pp. 837–845. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611974348.94

[123] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, 2001.

[124] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. New York, NY, USA: Cambridge University Press, 2011.

[125] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks: A unified approach to action segmentation," *ECCV Workshop on Brave New Ideas for Motion Representations in Video (BNMW)*, 2016.

[126] S. Stefati, N. Cowan, and R. Vidal, "Learning shared, discriminative dictionaries for surgical gesture segmentation and classification," in *MICCAI Workshop: M2CAI*, 2015.

[127] K. Pearson, "Note on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, no. 347-352, pp. 240–242, 1895.

[128] A. Malpani, S. S. Vendula, C. C. G. Chen, and G. D. Hager, "A study of crowdsourced segment-level surgical skill assessment using pairwise rankings," *International Journal of Computer Assisted Radiology and Surgery (IJCARS)*, 2012.

[129] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[130] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *arXiv preprint arXiv:1503.04069*, 2015.

[131] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2015. [Online]. Available: http://jmlr.org/proceedings/papers/v37/chung15.html

[132] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[133] J. Davis and A. Bobick, "The representation and recognition of action using temporal templates," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1997.

[134] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference Learning Representations (ICLR)*, 2014.

[135] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[136] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[137] V. Belagiannis, C. Rupprecht, G. Carneiro, and N. Navab, "Robust optimization for deep regression," in *International Conference on Computer Vision (ICCV)*. IEEE, December 2015.

[138] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014, pp. 1653–1660.

[139] T. Pfister, K. Simonyan, J. Charles, and A. Zisserman, "Deep convolutional neural networks for efficient pose estimation in gesture videos," in *Computer Vision–ACCV 2014*. Springer, 2015, pp. 538–552.

[140] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Neural Information Processing Systems (NIPS)*, 2015.

[141] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[142] L. Zappella, B. B. Haro, G. D. Hager, and R. Vidal, "Surgical gesture classification from video and kinematic data," *Medical Image Analysis*, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.media.2013.04.007

[143] C. Rupprecht*, C. Lea*, F. Tombari, N. Navab, and G. Hager, "Sensor substi-

tution for video-based action recognition," in *IEEE Conference on Intelligent Robots and Systems (IROS)*, 2016.

[144] Z. Bolei, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," 2015.

[145] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[146] W. Duch, J. Kacprzyk, E. Oja, and S. Zadrozny, Eds., *Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005, 15th International Conference, Warsaw, Poland, September 11-15, 2005, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 3697. Springer, 2005.

[147] V. Badrinarayanan, A. Handa, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," *arXiv preprint arXiv:1505.07293*, 2015.

[148] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *ICLR*, 2016.

[149] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," A. Waibel and K.-F. Lee, Eds.

San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 393–404. [Online]. Available: http://dl.acm.org/citation.cfm?id=108235.108263

[150] B. Ni, X. Yang, and S. Gao, "Progressively parsing interactional objects for fine grained action detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[151] A. Richard and J. Gall, "Temporal action detection using a statistical language model," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[152] S. Singh, C. Arora, and C. V. Jawahar, "First person action recognition using deep learned descriptors," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[153] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[154] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[155] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[156] B. Singh, T. K. Marks, M. Jones, O. Tuzel, and M. Shao, "A multi-stream bi-directional recurrent neural network for fine-grained action detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[157] H. Kuehne, A. Arslan, and T. Serre, "The language of actions: Recovering the syntax and semantics of goal-directed human activities," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[158] Y. Q. Limin Wang and X. Tang, "Action recognition with trajectory-pooled deep-convolutional descriptors," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[159] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," *CoRR*, vol. abs/1606.05328, 2016. [Online]. Available: http://arxiv.org/abs/1606.05328

[160] D. Klakow and J. Peters, "Testing the correlation of word error rate and perplexity," *Speech Communication*, vol. 38, no. 12, pp. 19 – 28, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167639301000413

[161] S. Franke, J. Meixensberger, and T. Neumuth, "Intervention time prediction from surgical low-level tasks." *Journal of Biomedical Informatics*, 2013.

[162] R. Stauder, V. Belagiannis, L. Schwarz, A. Bigdelou, E. Söhngen, S. Ilic, and N. Navab, "A User-Centered and Workflow-Aware Unified Display for the Operating Room."

[163] S. B. Deal, T. S. Lendvay, M. I. Haque, T. Brand, B. Comstock, J. Warren, and A. Alseidi, "Crowd-sourced assessment of technical skills: an opportunity for improvement in the assessment of laparoscopic surgical skills," *The American Journal of Surgery*, pp. 398–404, 2016.

[164] R. Kumar, A. S. Jog, A. Malpani, B. P. Vagvolgyi, D. D. Yuh, H. Nguyen, G. D. Hager, and C. Chen, "Assessing system operation skills in robotic surgery trainees," *The International Journal of Medical Robotics and Computer Assisted Surgery*, 2012.

[165] "TUM EndoVis," http://endovissub-workflow.grand-challenge.org/, 2015.

[166] O. Dergachyova, D. Bouget, A. Huaulmé, X. Morandi, and P. Jannin, "Auto-

matic data-driven real-time segmentation and recognition of surgical workflow," *International Journal of Computer Assisted Radiology and Surgery (IJCARS)*, 2016.

[167] F. Lalys, L. Riffaud, X. Morandi, and P. Jannin, "Automatic phases recognition in pituitary surgeries by microscope images classification," in *International Conference on Information Processing in Computer-Assisted Interventions (IPCAI)*. Geneve, Switzerland: Springer-Verlag Berlin, Heidelberg, Jun. 2010, pp. 34–44. [Online]. Available: http://www.hal.inserm.fr/inserm-00616977

[168] N. Padoy, D. Mateus, D. Weinland, M.-O. Berger, and N. Navab, "Workflow monitoring based on 3d motion features," in *International Conference on Computer Vision Workshops*, Sept 2009, pp. 585–592.

[169] R. Stauder, A. Okur, L. Peter, A. Schneider, M. Kranzfelder, H. Feussner, and N. Navab, "Random Forests for Phase Detection in Surgical Workflow Analysis," in *International Conference on Information Processing in Computer-Assisted Interventions (IPCAI)*, 2014.

[170] G. Quellec, K. Charriere, M. Lamard, Z. Droueche, C. Roux, B. Cochener, and G. Cazuguel, "Real-time recognition of surgical tasks in eye surgery videos," *Medical image analysis*, vol. 18, no. 3, pp. 579–590, 2014.

[171] A. Graves, S. Fernndez, and F. Gomez, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *In*

Proceedings of the International Conference on Machine Learning, ICML 2006, 2006, pp. 369–376.

Vita

Colin Lea is finishing his Ph.D. in Computer Science at Johns Hopkins University where he works on fine-grained action analysis for applications in robotics, surgery, and industrial manufacturing. He received his B.S. in Mechanical Engineering at the University at Buffalo Honors College in 2011 where his work ranged from computer vision systems for autonomous ground vehicles to haptic interaction. Colin was a National Science Foundation Graduate Research Fellow from 2012 to 2015 and an Intuitive Surgical Research Fellow from 2011 to 2012. As a graduate student he led the JHU Robo Challenge, an outreach effort for local middle and high schoolers, from 2013 to 2016. Colin will join Oculus Research in Pittsburgh after completing his Ph.D.