Parsing videos of actions with segmental grammars

Hamed Pirsiavash
Massachusetts Institute of Technology
hpirsiav@mit.edu

Deva Ramanan
University of California, Irvine
dramanan@ics.uci.edu

Abstract

Real-world videos of human activities exhibit temporal structure at various scales; long videos are typically composed out of multiple action instances, where each instance is itself composed of sub-actions with variable durations and orderings. Temporal grammars can presumably model such hierarchical structure, but are computationally difficult to apply for long video streams. We describe simple grammars that capture hierarchical temporal structure while admitting inference with a finite-state-machine. This makes parsing linear time, constant storage, and naturally online. We train grammar parameters using a latent structural SVM, where latent subactions are learned automatically. We illustrate the effectiveness of our approach over common baselines on a new half-million frame dataset of continuous YouTube videos.

1. Introduction

We focus on the task of action classification and segmentation in continuous, real-world video streams. Much past work on action recognition focuses on classification of pre-segmented clips. However, this ignores the complexity of processing video streams with an unknown number of action instances, each with variable durations and start/end times. Moreover, actions themselves exhibit internal temporal structure. For example, a 'making tea' action takes several minutes and requires multiple sub-actions such as heating water, preparing tea leaves, steeping the tea, and pouring into a cup. Each sub-action can vary in duration and sometimes temporal ordering.

Our work: In this work, we develop algorithms that report back *hierarchical parses* of long video streams at multiple temporal scales (Fig. 1). Our algorithms are based on grammars that compose videos out of action segments, and recursively compose actions out of subaction segments. We describe specialized grammars that can be parsed in an online fashion with a finite state machine. Our algorithms scale linearly with the length of a video and operate with bounded memory, crucial for processing streaming video sequences. Importantly, we describe methods for learning grammars from partially-labelled data. We assume training videos are provided with action labels, and describe a max-margin learning algorithm for latently inferring subaction structure.

Evaluation: Most datasets for activity recognition consist of pre-segmented clips. We have constructed a dataset of continuous temporal streams from YouTube sports clips. Our 5-hour video dataset contains continuous video streams with multiple types of actions with subaction structure. We will make this available to the community to spur further research.

2. Related Work

We refer the reader to the recent surveys of [20, 1] for a complete treatment of the large body of related work on action recognition. We focus on methods most related to our approach.

Spacetime templates: Many approaches to action recognition are based on spatiotemporal templates [11, 13, 21] defined on various features including optical flow [19, 4, 24] and bag-of-feature histograms built on space-time "words" [31, 16]. We use bag-of-word features as our data models.

Latent temporal structure: Inspired by [16, 28], we model action templates as compositions of subactions. For example, we learn that some weightlifting actions should be modeled as `yank`ing subactions followed by a `press`ing subaction (Fig. 1). However, most past work evaluates such models on single-action video clips. Our work differs in that we use grammars to compose multiple action instances together to yield a globally-optimal video parse.

Action segmentation: To process long video streams with multiple actions, one must address a temporal segmentation problem. Historically, this is most often done with a hidden markov model (HMM). Early approaches date back to finite state machine models [33, 17], while more recent work has explored discriminative variants such as CRFs [32]. There exists a large body of literature on extensions to HMMs in which states generate variable-length sequences; these are sometimes called hierarchical HMMs [5, 30, 17]

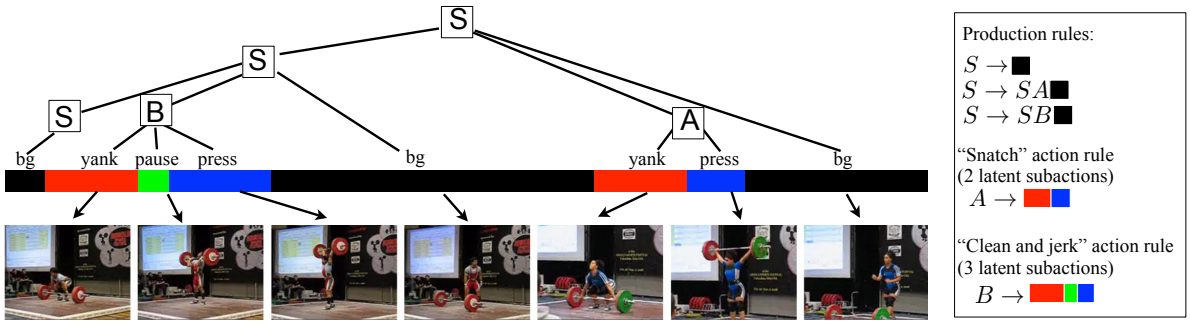

Figure 1. On the **left**, we show a hierarchical parse of a video segmented into actions (Snatch, Clean-and-Jerk) and sub-actions (yank, pause, press, background). Actions are represented by non-terminal symbols (N,C) while subaction and background segments are represented by terminal symbols (colored blocks). We show the associated grammar on the **right**.

or semi-Markov / segmental HMMs [8, 23]. Most related to our work are [25, 6], who use a semi-Markov model to segment video streams into actions. Our grammars do so while simultaneously parsing actions into subactions.

Grammars: Much prior work has explored context-free-grammars (CFGs) for gesture and event recognition [7, 15, 22, 26]. Attribute grammars [10] and interval logics [29, 12, 2] generalize CFGs to include context-sensitive constraints, typically with the cost of more expensive inference. Because of this burden, much prior work applies a grammar on a sparse set of primitive actions detected in a pre-processing stage. This makes inference scale with the number of detections rather than the length of the video. We describe hierarchical grammars that model actions and subactions with a finite-state machine. This allows us to efficiently process all frames in a video directly using our grammar.

3. Grammar model

Our primary contribution is a segmental regular grammar of actions, described in Section 3.4. To derive it , we first review the CYK parsing algorithm for general CFGs [14], as it forms the basis for our efficient parsing algorithm. One notable aspect of our grammar is that it generates variable-length segments instead of single tokens; to derive this feature, we first describe a simple modification to a CYK parser for handling such production rules.

3.1. Context-free grammars

A weighted CFG in Chomsky Normal Form (CNF) is specified by:

1. V is a finite set of non-terminal symbols
2. Σ is a set of terminal symbols
3. R is a set of rules of the form $X \rightarrow YZ$ or $X \rightarrow w$ for $X, Y, Z \in V$ and $w \in \Sigma$. Each rule $r \in R$ has an associated score $s(r, i, k, j)$ for instantiating it at boundary points i, j with a transition point of k.

A general CFG contains rules of the form $\alpha \rightarrow \beta$, where α is any nonterminal and β is any string of terminals and nonterminals. We write n_V for the number of non-terminal symbols, and n_R for the number of rules. One can show that any CFG can be written in CNF form by adding new rules with "dummy" nonterminals.

Given a sequence $w_1, \ldots, w_N$, the CYK algorithm is a $O(n_R N^3)$ dynamic programming algorithm for computing the best-scoring parse [14]. The algorithm will compute a table of partial parses, of size $O(n_V N^2)$. CYK explicitly computes the best parse of each possible segment and each possible symbol label for that segment. The key quantity which is iteratively computed is $\pi[X, i, j]$, the score of the best parse of the segment starting at frame i, ending at frame j, and labeled as symbol $X \in V$.

In the CYK algorithm, we first initialize the "bottom" row of the table, which represents the best parse of each one-frame-long segment:

$$\pi[X, i, i] = \max_{r \in \{X \rightarrow w\}} s(r, i) \qquad \text{for} \quad i = 1 \ldots N \qquad (1)$$

We can now populate the "second" row of the table, which represents the optimal parses of 2-frame segments. For a l-frame segment, we will look at all possible $l - 1$ splits and all possible n_r production rules that could generate this segment. Each one can be scored by looking at lower entries in the table, which have already been computed. We then take the max, and update the entry for the current l-long segment. We formally describe the algorithm in Alg. 1 and visualize the core loop in Fig. 2.

3.2. Segmental context-free grammars

In this section, we describe an extension to CYK-parsing that allows for production rules that generate multiple terminals. Though our extension is somewhat straightforward, we have not seen it derived in the literature and include it for completeness. We later show that segment-level production rules are crucial for capturing constraints on the duration of segments. Consider a set of CNF rules augmented with

```
1 for l = 2 : N do
2     for i = 1 : N − l + 1 do
3         j = i + l − 1;
4         π[X, i, j] =
              max_{r∈{X→YZ}, k∈{i...j−1}} s(r, i, k, j) + π[Y, i, k] + π[Z, k + 1, j];
5     end
6 end
```

Algorithm 1: CYK parsing. The parser iterates over segments of length l, and start positions of each segment i. For each of such N^2 segments, the parser searches over all rules (at most n_R) and split points (at most N) which could derive it (Fig. 2). This makes the overall algorithm $O(n_R N^3)$. For each of the N^2 table entries, one must store the score of the n_V possible symbols, making storage $O(n_V N^2)$.

those that allow nonterminals to generate a k-long segment of terminals:

$$X \to w_{1:k} \quad \text{where} \quad w_{1:k} = w_1 \dots w_k \tag{2}$$

Each of the above rules has an associated score $s(X \to w, i, j)$ for placing a $(k = j - i + 1)$-element segment starting at position i and ending at j. We call such a CFG a segmental-CFG (SCFG). SCFGs can be seen as a type of CFG where the *number of rules scales with* N, complicating inference. An alternative approach would be to encode segment length k as an attribute in an attribute grammar. However, the resulting constraints on production rules are context sensitive, and so are no longer parsable with CYK [27].

We show that, with a simple modification, CYK can accept rules of the form (2), keeping the algorithm at $O(n_R N^3)$ without *any* increase in complexity. Replace Line 4 of CYK with the following two lines:

$$v = \max_{\substack{r \in \{X \to YZ\} \\ k \in \{i \dots j-1\}}} s(r, i, k, j) + \pi[Y, i, k] + \pi[Z, k+1, j]$$

$$\pi[X, i, j] = \max_{\substack{r \in \{X \to w_{1:k}\} \\ k = (j-i+1)}} \Big(v, s(r, i, j)\Big) \tag{3}$$

For each table entry, we search over derivations as before (equivalent to Line 4 of CYK), but now we also search over segmental terminal derivations from the lowest layer. We need check for only $(k = j - i + 1)$-long terminal derivations, of which there exist at most n_R. This means that one can parse a segmental CFG in $O(n_R N^3)$.

3.3. Regular grammars

CYK parsing may be difficult to apply to long videos since computation and storage grows super-linearly with the length of the video. Chomsky describes a special case of context-free-grammars known as finite-state grammars or regular grammars [3], which consist of restricted rules with a single non-terminal followed by any number of terminals on the right-hand side:

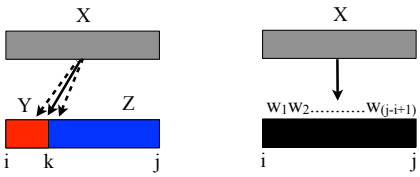

Figure 2. To compute the score of segmenting and labeling frames i to j as symbol X, CYK searches over all production rules $(X \to YZ)$ and split points k that could derive X (**left**). Our segmental CYK parser augments the above search by also considering derivations that directly score the observed terminals/features (**right**).

$$X \to Yw, \qquad X \to w \tag{4}$$

Such grammars cannot model recursively-defined languages, such as strings of nested parentheses [14]. However, they can be parsed with finite state machines by reducing them to first-order conditional Markov models with the following steps: add dummy nonterminals to convert production rules to CNF (if they are not already), define a markov state for each non-terminal symbol, and finally define a transition from states Y to X for each production rule. Such grammars can be parsed with a standard Viterbi decoder in $O(n_R N)$, where n_R (the number of transitions) is upper bounded by n_V^2 [5]. Hence regular grammars naturally take advantage of sparse Markov transition matrices.

Regular grammars are widely used to specify regular expressions used in pattern matching [14]. We describe a regular grammar for parsing weightlifting videos in Fig. 1. For example, a clean-and-jerk action (B) is defined by a "string" of `yank`, `pause`, and `press` terminals. The regular grammar is expressed in CNF form in Fig. 4. Though such long-range constraints can be expressed in a Markov model with augmented states (corresponding to dummy nonterminals) and sparse transitions, production rules with regular expressions are a more concise representation.

3.4. Segmental regular grammars

We define a segmental RGs (SRGs) to be a RG with production rules that generate arbitrary length sequences of terminals:

$$X \to Yw_{1:k}, \qquad X \to w_{1:k} \tag{5}$$

Note that SRGs can be converted to RGs by adding dummy nonterminals. However, we show that one can devise a parser that directly handles the segmental rules above. Our SRG parser can either be seen as a special case of segmental CYK parsing or an extension of Viterbi decoding.

Restricted CYK: Assume scores can be written as $s(X \to Yw, i, j)$, where i is the start and j is the end of the $(k = j - i + 1)$-element segment. Assume that segments can be at most L elements long. SRGs can be parsed

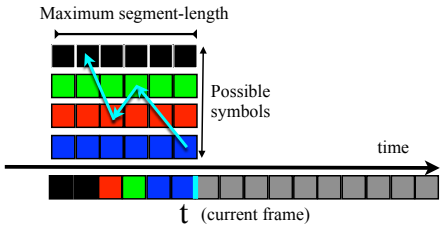

Figure 3. We visualize a run of our SRG parser from Alg. 2. We label symbols with the terminal color used to derive them, and use blue arrows to denote argmax pointers from Line 2 of Alg. 2. In the text, we show that our parser can be seen as a special case of CYK and Viterbi decoding for sparse semi-Markov models.

in $O(n_R NL)$, where n_R is the number of rules in the underlying non-segmental grammar. The algorithm is similar to CYK except that we need to maintain only the first column of table entries $\pi[X, i, j]$ for a fixed $i = 1$. We write this as $\pi[X, j]$, the score of the best parse from position 1 to j, given that the symbol at j is X. For notational simplicity, let us define an empty symbol $\{\}$ which allows us to write both rules from (5) using a single form. Initialize $\pi[\{\}, 1] = 0$ and $\pi[\{\}, j] = -\infty$ for $j > 1$:

1 **for** $j = 1 : N$ **do**

2 $\quad \pi[X, j] = \max_{\substack{k \in \{1 \ldots L\} \\ r \in \{X \to Yw\}}} \pi[Y, j-k] + s(r, j-k+1, j)$

3 **end**

Algorithm 2: Our SRG parser, which is visualized in Fig.3. The inner loop searches for best derivation of X that transitions from a previous nonterminal Y, at some position $j - k$ (after which the k remaining elements are segmented into X). At each iteration, the algorithm searches over L possible transition points and n_R possible rules for transitioning at that point, making overall computation $O(n_R NL)$.

Online parsing: If we interpret j as indexing time, the algorithm is naturally online. At any point, we can compute the score of the best parse found until now with $\max_X \pi[X, j]$. Storage for online parsing is $O(n_V L)$ – independent of N – because we only need to store table entries for the past L timesteps. This makes it scalable to long (even streaming) data. We visualize a run of our parser in Fig. 3.

Sparse markov decoding: Just as RGs can be written as Markov models with sparse transition between states, SRGs can be written as semi-Markov models with sparse transitions between segment labels. Define a Markov segment label corresponding to each nonterminal symbol, and define the score of rule $s(X \to Yw_{1:k}, j)$ is the score of transitioning from segment label Y into segment label X, where frames from $j - k + 1$ to j are labeled with X [23]. SRGs are useful because they make use of concise regular expressions to encode long-range constraints between segments.

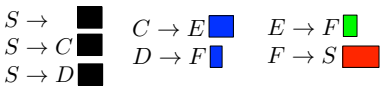

Figure 4. The SRG rule-set used to generate the actual parse shown in Fig. 1, where S is the sentence symbol corresponding to the full parse till now, C-F are abstract symbols, and colored boxes are variable length terminals representing yank, pause, and press subactions. A black box is the background terminal. To write the grammar in CNF form, we introduce symbols C and D corresponding to parses ending in a completed "snatch" (SA) and "clean-and-jerk" (SB) actions respectively, and E and F corresponding to parses ending in incomplete actions. Each subaction is parametrized by a data model α (visualized with different colors) and its ideal temporal length β (visualized with boxes of different lengths), as in (7).

Parse tree: We can recover the optimal parse tree by storing the argmax pointers from Line 2 of the algorithm. Let $R[X, j]$ and $K[X, j]$ point to argmax rule r and offset k. We can represent a parse tree with a collection of (r, j) pairs obtained by backtracking through R and K, initializing with the highest-scoring symbol at the last frame. We visualize these backtracked argmax pointers as blue arrows in Fig. 3. We encode a parse-tree as

$$P = \{(r_m, j_m) : m = 1 \ldots M\}$$

where M refers to the number of distinct segments in the parse, j_m is the last frame of the m^{th} segment, r_m is the rule id whose right-hand-side instanced that segment.

4. Model structure

We model all actions with production rules of the form:

$$A \to xyz$$

where A is an action and x, y, z are variable-length subactions. We select the optimal number of subactions through cross-validation, but find 3 to work fairly consistently. Our SRG grammar is of the form:

$$S \to b, \quad S \to SAb, \quad S \to SBb, \quad \ldots$$

where S is a valid parse and b is a variable-length background terminal. The background may have a large variation, so we can introduce multiple background terminals (multiple mixtures) and let the parser choose the best one. In practice, we find that a single background terminal works fairly well. Note that segment-level production rules are crucial since they capture global features within segments like the duration of segments.

Given a data sequence D and a candidate parse $P = \{(r_m, j_m)\}$, we can write its score under our weighted SRG grammar as

$$S(D, P) = \sum_{m=1}^{M} s(D, r_m, i_m, j_m) \tag{6}$$

where $i_m = j_{m-1} + 1$ and we explicitly denote the dependence of each segment's score $s(r_m, i_m, j_m)$ on data D. To further describe our particular data model, let us define x_m to be the terminal on the right-hand-side of rule r_m:

$$s(D, r_m, i_m, j_m) = \alpha_{x_m} \cdot \phi(D, i_m, j_m) + \beta_{r_m} \cdot \psi(k_m) + \gamma_{r_m} \quad (7)$$

To describe the parameters of our model, we will use an illustrative example of a grammar defined for weightlifting actions, shown in Fig. 1 and defined in Fig. 4. Terminal symbols correspond to subactions, while abstract symbols correspond to actions. A "clean-and-jerk" action is defined by a yank, pause, and press subaction, while a "snatch" action is defined by a yank and press.

Data model: The first term is a data term that extracts features from segment spanning frames i_m to j_m, and scores them with a model tuned for subaction (terminal) x_m. We compute a bag-of-features descriptor ϕ for each segment, where a feature is a space-time visual word [31, 28]. We can interpret α_{x_m} as a model tuned for particular subactions; for example, the model for $x_m =$ yank will be tuned for spacetime words that fire on crouching poses. Note that α is symbol-specific x_m rather than rule-specific r_m; this allows different rules to share data models for equivalent symbols. For example, the "clean-and-jerk" and "snatch" actions in Fig. 4 share the same subaction data models.

Temporal model: The second term is analogous to a temporal "prior" that favors particular segment lengths $k = j_m - i_m + 1$ for a subaction. β is rule-specific r_m and not symbol-specific x_m. This allows a yank subaction to have different priors for its length depending on which action it was instanced. Specifically, we define $\psi(k_m) = \begin{bmatrix} k_m & k_m^2 \end{bmatrix}$. This means that the parameters β_{r_m} can be interpreted as the rest position and rigidity of a spring that defines the ideal length of a segment. In our experimental results, we show that such temporal terms are vital to ensuring good segmentation accuracy. Furthermore, such constraints are difficult to encode by a standard HMM.

Rule model: The last term γ_{r_m} is a scalar "prior" or bias that favors the use of certain rules over others. This may encode the fact that "clean-and-jerk"s are more likely to occur than "snatch" actions, for example.

5. Learning

Our model from (7) is linear in the model parameters $w = \{\alpha, \beta, \gamma\}$, allowing us to write the score of a parse as a linear function $S(D, P) = w \cdot \Phi(D, P)$. Though we learn subactions in a latent framework, we initially describe the fully-supervised scenario for ease of exposition.

Fully-supervised learning: Assume we are given training data of videos with ground-truth parses $\{D_n, P_n\}$ and a manually-specified set of production rules Γ. We wish to learn weights w for the data model α, temporal model β, and rule compositions γ. The weights should score correct parses highly and incorrect parses poorly; we do this by defining a structured prediction learning function

$$\arg \min_{w, \xi_n \geq 0} \quad \frac{1}{2} w \cdot w + C \sum_n \xi_n \qquad \text{s.t.} \quad \forall n, \forall H_n \quad (8)$$
$$w \cdot \Phi(D_n, P_n) - w \cdot \Phi(D_n, H_n) \geq \text{loss}(P_n, H_n) - \xi_n$$

The above linear constraints state that, for each training video D_n, the true parse P_n should outscore a hypothesized parse H_n by some amount given by $\text{loss}(P_n, H_n)$. We experiment with different loss functions. Many approaches use a Hamming loss, which simply counts up the number of frames with incorrect symbol labels. We also considered a simpler 0-1 loss that defines a candidate parse as incorrect if none of its transitions are near ground-truth transitions. Finally, we allow these constraints to be loosely satisfied using slack variables. The above is equivalent to a structural SVM, for which many well-tooled solvers exist [9]. The main computational step in learning requires solving a "loss-augmented" inference problem, where for each training example, one finds the worst-offending parse:

$$H_n^* = \max_H w \cdot \Phi(D_n, H) + \text{loss}(P_n, H) \quad (9)$$

Our loss functions can be absorbed into the data term of (7), implying that our efficient SRG parser can be used to find such an offensive parse.

Latent learning of subactions: Specifying a full parse tree in learning can be expensive or ambiguous. In our scenario, we are given video streams with action labels for each frame, without any subaction labels. We denote the action labels for video n as A_n. In this setting, we automatically estimate subactions with a latent structural SVM [34]. We use the CCCP algorithm [35] by iterating between the following:

1. Given a model w, infer a parse P_n for each video n consistent with A_n.

2. Given a set of full parses $\{P_n\}$, learn a model w by solving the QP of (8) with an action-specific loss.

Step 1 is implemented by modifying the loss function (9) to penalize non-consistent parses with an infinite loss. This loss also can be absorbed into the data term. In Step 2, we define an action-specific $\text{loss}(A_n, H)$ that penalizes disagreement of action transitions rather than subaction transitions. Though data models, temporal priors, and rule weights are fully learned, we must still specify the production rules themselves. Additionally, we must specify a starting state for the above iteration. Both are described below.

Initialization: Our grammar can encode rules between action categories. However, in our experiments, we analyze videos that contain instances of a single action category. For simplicity, we learn separate grammars for each action category using the following generic rule set: $\{S \rightarrow b, S \rightarrow SAb, A \rightarrow xyz\}$. We initialize the iterations by segmenting each action instance A (in the training set) into a fixed number of equally-sized subactions xyz. We use cross-validation to select the optimal number of subactions per action class, though we find 3 to work fairly consistently. The final inferred subaction labels (latently estimated on both training and testing data) can be quite non-uniform, as shown in Fig. 6 and 7. Our latent learning scheme is similar to the work of [28], except that we learn rules for globally parsing a video stream with multiple actions rather than a single isolated action.

6. Experiments

Lack of data: Popular action benchmarks such as the MED challenge in TRECVID consist of short video clips with single actions, and so are not appropriate for evaluating our method. Previous action segmentation methods are often evaluated on artificially-constructed videos, obtained by concatenating together pre-segmented video clips [6, 25]. This is clearly limiting. Moreover, we are not aware of any benchmark video dataset consisting of continuous unscripted human actions, outside of surveillance footage [1] and wearable camera recordings [18].

Continuous Olympics Dataset: To address this deficiency, we have collected our own dataset inspired by [16], which introduced a dataset of amateur and realistic (but pre-segmented) YouTube video clips of 16 different Olympic sports. We have collected continuous videos of a subset of 8 actions from YouTube. Our Continuous Olympics dataset contains almost 5 hours or half a million frames of such realistic video footage. Each video contains an average of 6 action instances, each annotated with start/end frame labels. Fig. 5 illustrates a sample frame of each action. We will release this dataset to spur further research in real-world action parsing. We use a 50-50 train/test split in our experiments.

Evaluation: We assume we are given a video of a known action category, and then run our action-specific grammar to parse the video into actions and subactions. Because we do not have groundtruth subaction labels, we evaluate only action label accuracy. We show qualitative results in Fig. 6 and 7, and include parsed videos in our supplementary material. We refer the reader to figure captions for a detailed analysis. Given a candidate parsing of a sequence, we quantitatively evaluate it (Fig. 8) in two ways. **Per frame**: We count the number of frames with matching candidates and ground truth action labels. **Action detection:** Similar to PASCAL evaluation of object detection, we think of our parser as returning candidate action-segment "detections". A detection is considered a true positive if the overlap (union/intersection) between it and the ground-truth action segment is more than a predefined threshold. We use a default of 40%, as we found it consistent with visual inspection. We also evaluate other thresholds.

Figure 5. Our Continuous Olympics action dataset contains video streams of 8 different actions ("weightlifting", "javelin", "long-jump", "vault", "bowling", "diving", "hammer-throw", "tennis") collected from YouTube. There is large variety in view point, scale, duration, and the number of action instances in each video.

Baselines: We saw poor performance for simplistic baselines such as a scanning window template and single-frame HMM. We compare against the recent state-of-the-art model of [28], who learn an action-specific **subaction** model with 3 subactions. Actions are modeled using a semi-Markov model, where states correspond to latently-inferred subactions. However, it is not clear how to use this model to detect multiple actions in a single video. We apply it in a scanning window fashion, and output a set of non-overlapping action segments with NMS. This produces reasonable frame labels (36% correct), but struggles at segment detection (2% AP). This indicates the difficulty of segment detection. We also compare to the **segmental action** model of [25], which uses a semi-Markov model to process an entire video, where states correspond to action / background labels with priors over temporal duration. Such models were also explored in [6], when trained with alternative objective functions. Algorithmically, such models correspond to a "one-level" version of our grammar without subactions. They directly produce a global parse without requiring NMS, but do not reason about subactions. This improves accuracy to 15% AP.

Our model: Our final model can be seen as a combination of [28] and [25] that makes use of an augmented state space (defined by our production rules) to reason about both actions and subactions. Our final accuracy is 22% AP. Because our parser is algorithmically equivalent to a semi-Markov model with sparse transitions, it is essentially as fast as both baselines. We also compare to a version of our model with **no length prior** on the duration of a subaction. This drastically drops performance from 22% to 8%. Finally, we use a segmental CYK parser (3) to parse our SRG, which produces identical results (as excepted). However, our parser take roughly .7 ms/frame (similar to our baselines), while CYK takes roughly 2.3 sec/frame (1000X

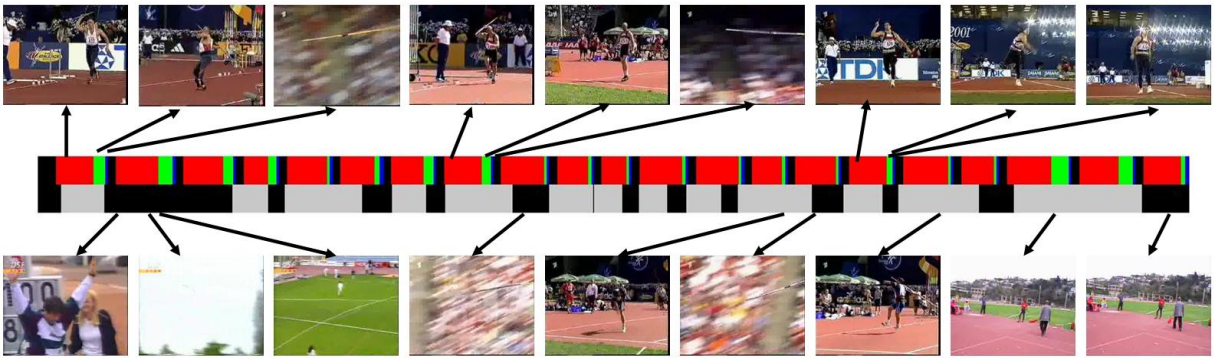

Figure 6. A test video containing 'javelin throw" actions. The bottom timeline shows ground-truth action labels (in gray), while the top timeline shows inferred action and sub-action labels. The learned grammar infers three subactions loosely corresponding to running, release, and throwing. The release sub-action is short and is not always visible in the figure. Our grammar model is able to enforce the presence of such short but crucial sub-actions.

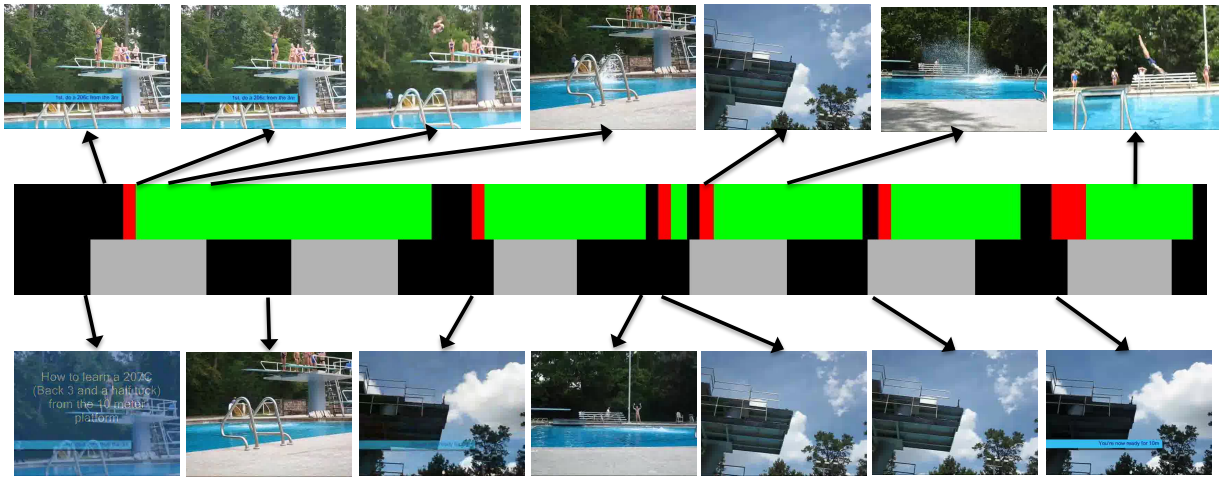

Figure 7. A test video containing 'diving" actions, where ground-truth action labels are shown in gray. We latently infer 2 subactions loosely correspond to initial bending and jumping. Some misalignment errors are due to ambiguities in the ground-truth labeling of action boundaries. Overall, our parser produces a reasonable estimated count of action instances (though the first two action instances are over-merged into one action segment.)

slower). Note that CYK can handle more general CFGs.

Analysis: Our subaction model performs best on actions with clear structure, such as weightlifting, pole-vaulting, and diving. Sometimes subactions can be semantically interpretable, but this need not be case as the structure is latently inferred. Overall, our model is reasonably successful at labeling frames (62% accuracy) but still finds segment detection challenging (22% AP). Reducing the overlap threshold (for labeling an action segment as correct) from 40% to 10% increases AP from 22% to 43%. Ground-truth labeling of action boundaries can be ambiguous, and so lower overlap thresholds are reasonable for evaluation. Moreover, low thresholds still score the ability to *count* the number of action instances in a video. Our results suggest that our parser can be used for fairly accurate counting of action instances.

Conclusion: We have described segmental extensions of grammars for action-parsing, focusing on efficient segmental regular grammars. We show that such models capture temporal constraints at multiple scales, both between actions and between subactions. We introduce parsing algorithms that are linear-time, constant memory, and online, and so quite practical for long-scale videos. We also described max-margin algorithms for inferring latent subactions form partially-labeled training data. To illustrate our approach, we introduced a novel dataset of continuous actions and demonstrated encouraging performance over a number of standard baseline approaches.

Acknowledgements: Funding for this research was provided by NSF Grant 0954083, ONR-MURI Grant N00014-10-1-0933, and the Intel Science and Technology Center - Visual Computing.

References

[1] J. K. Aggarwal and M. S. Ryoo. Human activity analysis: A review. *ACM Comput. Surv.*, 43(3):16, 2011.

[2] W. Brendel, A. Fern, and S. Todorovic. Probabilistic event logic for interval-based event recognition. In *CVPR*, 2011.

[3] N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 1956.

Segment Detection (average precision)/ Frame Labeling (% frames)

Action name	Subactions [28]	our model (no length prior)	Segmental actions [25]	our model
weightlifting	0.20 / 0.51	0.13 / 0.74	0.27 / 0.6	0.53 / 0.6
javelin	0.006 / 0.54	0.11 / 0.64	0.36 / 0.59	0.36 / 0.65
long-jump	0 / 0.26	0.02 / 0.46	0.10 / 0.54	0.10 / 0.71
vault	0 / 0.28	0.12 / 0.49	0.06 / 0.34	0.11 / 0.69
bowling	0.007 / 0.39	0.08 / 0.45	0.21 / 0.51	0.25 / 0.54
diving	0 / 0.48	0.10 / 0.53	0.02 / 0.45	0.08 / 0.62
hammer-throw	0.006 / 0.25	0.04 / 0.36	0.16 / 0.29	0.23 / 0.63
tennis	0 / 0.15	0.01 / 0.37	0 / 0.37	0.08 / 0.52
Average	**0.027 / 0.36**	**0.076 / 0.51**	**0.15 / 0.46**	**0.22 / 0.62**

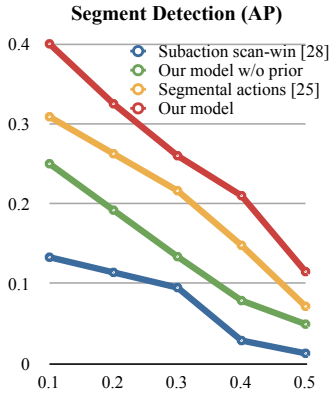

Figure 8. We compare our approach to various baselines, evaluating both action segment detection and frame labeling accuracy. On the **left** table, we use a segment overlap threshold of 40%, but examine performance for different thresholds on the **right** figure. Please see text for a discussion, but overall, our approach significantly outperforms past approaches based on subaction models [28] and segmental actions [25] because our grammar simultaneously captures both types of constraints.

[4] A. Efros, A. Berg, G. Mori, and J. Malik. Recognizing action at a distance. In *CVPR*, 2003.

[5] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine learning*, 32(1):41–62, 1998.

[6] M. Hoai, Z. Lan, and F. De la Torre. Joint segmentation and classification of human actions in video. In *CVPR*, 2011.

[7] Y. Ivanov and A. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE PAMI*, 2000.

[8] J. Janssen and N. Limnios. *Semi-Markov models and applications*. Kluwer Academic Publishers, 1999.

[9] T. Joachims, T. Finley, and C. Yu. Cutting plane training of structural SVMs. *Machine Learning*, 2009.

[10] S. Joo and R. Chellappa. Attribute grammar-based event recognition and anomaly detection. In *CVPR-W*, 2006.

[11] Y. Ke, R. Sukthankar, and M. Hebert. Event detection in crowded videos. In *ICCV*, pages 1–8. IEEE, 2007.

[12] S. Kwak, B. Han, and J. Han. On-line video event detection by constraint flow. *IEEE PAMI*, 2013.

[13] I. Laptev and P. Perez. Retrieving actions in movies. In *International Conference on Computer Vision*, 2007.

[14] C. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[15] D. Moore and I. Essa. Recognizing multitasked activities from video using stochastic context-free grammar. In *National Conf on Artificial Intelligence*, 2002.

[16] J. Niebles, C. Chen, and L. Fei-Fei. Modeling temporal structure of decomposable motion segments for activity classification. *ECCV*, pages 392–405, 2010.

[17] N. Oliver, A. Garg, and E. Horvitz. Layered representations for learning and inferring office activity from multiple sensory channels. *CVIU*, 96(2):163–180, 2004.

[18] H. Pirsiavash and D. Ramanan. Detecting activities of daily living in first-person camera views. In *CVPR*, 2012.

[19] R. Polana and R. Nelson. Detection and recognition of periodic, nonrigid motion. *IJCV*, 23(3):261–282, 1997.

[20] R. Poppe. A survey on vision-based human action recognition. *Image and Vision Computing*, 2010.

[21] M. Rodriguez, J. Ahmed, and M. Shah. Action mach a spatio-temporal maximum average correlation height filter for action recognition. In *CVPR*, pages 1–8, 2008.

[22] M. S. Ryoo and J. K. Aggarwal. Semantic representation and recognition of continued and recursive human activities. *IJCV*, 82(1):1–24, 2009.

[23] S. Sarawagi and W. Cohen. Semi-markov conditional random fields for information extraction. *NIPS*, 2004.

[24] E. Shechtman and M. Irani. Space-time behavior based correlation. In *IEEE PAMI*, 2007.

[25] Q. Shi, L. Cheng, L. Wang, and A. Smola. Human action segmentation and recognition using discriminative semi-markov models. *IJCV*, pages 1–11, 2010.

[26] Z. Si, M. Pei, B. Yao, and S. Zhu. Unsupervised learning of event and-or grammar and semantics from video. ICCV, 2011.

[27] K. Slonneger and B. L. Kurtz. *Formal syntax and semantics of programming languages*. Addison-Wesley, 1995.

[28] K. Tang, L. Fei-Fei, and D. Koller. Learning latent temporal structure for complex event detection. In *CVPR*, 2012.

[29] S. Tran and L. Davis. Event modeling and recognition using markov logic networks. *ECCV*, 2008.

[30] T. T. Truyen, D. Q. Phung, H. H. Bui, and S. Venkatesh. Hierarchical semi-markov conditional random fields for recursive sequential data. In *NIPS*, pages 1657–1664, 2008.

[31] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC*, 2009.

[32] S. Wang, A. Quattoni, L. Morency, D. Demirdjian, and T. Darrell. Hidden conditional random fields for gesture recognition. In *CVPR*. IEEE, 2006.

[33] A. Wilson and A. Bobick. Parametric hidden markov models for gesture recognition. *PAMI*, 21(9):884–900, 1999.

[34] C.-N. J. Yu and T. Joachims. Learning structural svms with latent variables. In *ICML*, 2009.

[35] A. L. Yuille, A. Rangarajan, and A. Yuille. The concave-convex procedure (cccp). *NIPS*, 2:1033–1040, 2002.