Budget-Aware Activity Detection with A Recurrent Policy Network

Supplementary Material

Behrooz Mahasseni[†1]
https://mahasseb.github.io
Xiaodong Yang[2]
http://xiaodongyang.org
Pavlo Molchanov[2]
pmolchanov@nvidia.com
Jan Kautz[2]
http://jankautz.com

[1] Apple Inc.
[2] NVIDIA Research

1 Estimation of Detection Time

It is challenging to decide the precise computation time of the different methods due to (1) lack of computational information in the literature, and (2) multiple complex stages involved. We adopt the following strategies to estimate the approximate computation time for them: (1) directly use the speed or time if it is reported in the related papers; (2) use the speed or time of our components (e.g., CNN and LSTM) to infer others; (3) use the processing bottleneck that dominates the computational costs (e.g., extraction of hand-crafted features) to approximate the overall time.

1.1 THUMOS14

There are 20 action classes and 1574 videos in the testing set of THUMOS14 [9]. Each video has 5507 frames on average.

- **Ours** is provided in the analysis of computational costs of Figure 3(b) in the main paper. The detailed computation time of each component is: forward pass of VGG16 is 3.0ms for each frame on the GPU; execution time of 2-layer LSTM at each step is 5.4ms on the GPU; pixel-level frame difference is 0.1ms per frame on the CPU; linear regression is 5.5ms on the CPU. We use 6 steps to run our policy for each video, 15 frames as the neighborhood of each selected frame, and 10 uniformly sampled frames for regression. So the overall computation time for each video is: $(0.1 \times 15 \times 6) + (3.0 \times 15 \times 6) + (5.4 \times 6) + (0.1 \times 10) + (3.0 \times 10) + 5.5 \approx 348\text{ms} \approx 0.35\text{s}$.
- **Glimpses** [29] downsamples videos by 5 (i.e., to 5fps) for this dataset, and observes 2% (or less) of video frames for inference. Since it is a binary model, we need to

[†] Work done during an internship at NVIDIA Research.

train and run 20 models to detect all classes of this dataset. Based on the time of our approach, we can infer the computation time of each component in [29]: forward time of VGG16 is 3.0ms for each frame on the GPU; execution time of 3-layer LSTM at each observation is 8.2ms on the GPU. So the average computation time to detect the whole classes for each videos is: $5507 \div 5 \times 0.02 \times (3.0+8.2) \times 20 \approx 4932$ms ≈ 4.9s.

- **R-C3D** [26] runs at 569fps on an NVIDIA Titan X Maxwell GPU and 1030fps on an NVIDIA Titan X Pascal GPU. So the average computation time to predict each video is: $5507 \div 569 \approx 9.7$s on Titan X Maxwell, and $5507 \div 1030 \approx 5.3$s on Titan X Pascal.

- **Language Model** [13] takes 7.5h on the CPU with eight 1.2GHz cores for inference on this dataset. So the average computation time for each video is: $7.5 \times 60 \times 60 \div 1574 \approx 17$s.

- **DAPs** [4] integrates C3D and LSTM, and runs at 134.1fps on the GPU. So the average computation time to predict each video is: $5507 \div 134.1 \approx 41$s.

- **S-CNN** [14] employs the proposal and localization networks for inference, and runs at 60fps on the GPU. So the average computation time for each videos is: $5507 \div 60 \approx 92$s.

- **CDC** [15] alone runs at 500fps on the GPU to refine predicted temporal boundaries. However, on this dataset, it relies on the proposals generated by S-CNN [14], which dominates the computational costs. So the overall computation time including segment proposal and prediction refinement for each video can be approximated by that of S-CNN: 92s.

- **Fast Temporal Proposal** [8] downsamples videos by 5 and runs at 10.2fps on the CPU. So the average computation time for each video is: $5507 \div 5 \div 10.2 \approx 108$s.

- **Pyramid of Scores** [30] employs a sliding window strategy, and requires time-consuming feature extraction, e.g., 21d to extract the iDT features on the CPU, 20h for feature encoding on the CPU, etc. So the average computation time to predict each video is much longer than the proposal based method in [8]: > 108s.

- **CUHK14** [22] is also a sliding window based approach. It extracts deep features by AlexNet, and iDT features running at 3.5fps on the CPU. Additional computations include FV encoding of iDT features and SVM classification. So the average computation time to process each video is also much longer than the proposal based method in [8]: > 108s.

- **LEAR14** [12] is also based on sliding windows with computationally expensive feature extractions: SIFT, iDT, color features, FV encoding, deep features by AlexNet, as well as acoustic features such as MFCC and ASR. So the average computation time to predict each video is also much longer than the proposal based method in [8]: > 108s.

1.2 ActivityNet

There are 200 activity classes and 4926 videos in the validation set of ActivityNet [7]. Each video has 3254 frames on average.

- **Ours** is independent of video length, but is determined by the number of steps to run our policy. We take the same processing as for THUMOS14 to perform activity detection on this dataset. So the overall computation time to predict each video is the same: $(0.1\times15\times6)+(3.0\times15\times6)+(5.4\times6)+(0.1\times10)+(3.0\times10)+5.5\approx$ 348ms $\approx$ 0.35s.

- **Glimpses** [29] downsamples videos by 25 (i.e., to 1fps) for this dataset, and observes 2% (or less) of video frames for inference. Similarly, we need to train and run 200 models to detect the whole classes of this dataset. So the average computation time to detect the entire classes for each video is: $3254\div25\times0.02\times(3.0+8.2)\times200\approx$ 5824ms $\approx$ 5.8s.

- **R-C3D** [26] runs at the same speed: 569fps on an NVIDIA Titan X Maxwell GPU and 1030fps on an NVIDIA Titan X Pascal GPU. So the average computation time to predict each video is: $3254\div569\approx5.7$s on Titan X Maxwell, and $3254\div1030\approx3.2$s on Titan X Pascal.

- **OBU16** [18] uses C3D for feature extraction running at 313.9fps on the GPU, a binary random forest, and a dynamic programming process for generating proposals. So only considering the feature extraction, the average computation time to process each video is: $3254\div313.9\approx10$s.

- **UPC16** [11] consists of C3D running at 313.9fps on the GPU, and a single layer LSTM predicting with 2.7ms on the GPU. C3D uses a short video clip of 16 frames as input. So the average computation time to predict each video is: $(3254\div313.9)$s $+$ $(3254\div16\times2.7)$ms $\approx$ 11s

- **UTS16** [23] extracts a variety of features such as iDT features, C3D features, deep features by ResNet152 (pre-trained on ImageNet), deep features by ResNet152 (pre-trained on Places2), and deep features by InceptionV3. Additional costs involve VLAD encoding of multiple features and SVM classification. So only considering the feature extraction by iDT which runs at 3.5fps on the CPU, the average computation time to process each video is: $3254\div3.5\approx930$s.

- **CDC** [15] relies on the detection outputs of UTS16 [23], i.e., it is used to refine the predicted results of UTS16. While CDC runs at 500fps in the refining process, the computation bottleneck comes from generating the temporal segments by UTS16. So the overall computation time is determined by UTS16: 930s.

2 Illustration of Policy Execution

Figure 5 illustrates the frame selection and prediction process of the learned policy. Each colored box above the frame sequence shows the predicted action class (with associated probability score), and detected temporal segment (from start to end). We can directly discard the segments that are predicted as background. At steps 2 and 3, the policy makes the true positive predictions that match to the two ground truth segments. Frames of the observing sequence present the selected frames for the corresponding steps.

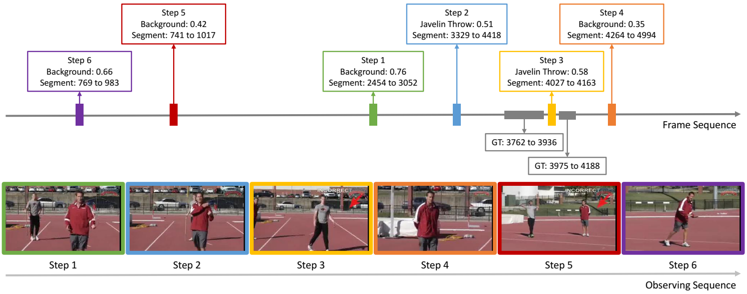

Figure 5: Illustration of the learned policy running for frame selection on THUMOS14.

3 More Results on ActivityNet

Since the glimpses method [29] is a binary model and their detection results on the entire 200 classes of ActivityNet are not provided (otherwise they have to train 200 models for each class), we train our policy on the same two subsets (i.e., sports and work) as [29] for fair comparisons. As shown in Tables 3 and 4 (IoU = 0.5), our approach perform better than [29] in 15 out of 21 classes on the sports subset, and 9 out of 15 classes on the work subset. Overall, we outperform [29] by 2.0% and 1.8% in mAP on the sprots and the work subsets, respectively. More importantly, we need only a single training phase to handle the entire activity classes of each subset, while [29] requires to train multiple models for every class.

Class	Glimpses [29]	Ours	Class	Glimpses [29]	Ours
Archery	5.2	**13.7**	Long Jump	**56.8**	48.7
Bowling	52.2	**52.4**	Mountain Climb.	**53.0**	52.4
Bungee	**48.9**	46.3	Paintball	12.5	**24.9**
Cricket	38.4	**39.1**	Playing Kickball	60.8	**61.2**
Curling	30.1	**32.3**	Playing Volleyball	**40.2**	39.2
Discus Throw	17.6	**21.8**	Pole Vault	35.5	**40.2**
Dodgeball	**61.3**	60.2	Shot Put	50.9	**51.4**
Doing Moto.	46.2	**47.3**	Skateboard.	**34.4**	32.7
Ham. Throw	13.7	**18.8**	Start Fire	38.4	**40.1**
High Jump	21.9	**27.4**	Triple Jump	16.1	**22.7**
Javelin Throw	35.7	**40.1**			
mAP				36.7	**38.7**

Table 3: Comparison of the per-class breakdown AP on the sports subset of ActivityNet.

Class	Glimpses [29]	Ours	Class	Glimpses [29]	Ours
Attend Conf.	**56.5**	53.8	Phoning	**52.1**	46.7
Search Security	33.9	**36.1**	Pumping Gas	34.0	**49.3**
Buy Fast Food	45.8	**48.2**	Setup Comp.	30.3	**35.1**
Clean Laptop	35.8	**38.3**	Sharp. Knife	35.2	**38.3**
Making Copies	**41.7**	39.5	Sort Books	16.7	**33.7**
Organizing Boxes	19.1	**24.5**	Using Comp.	**50.2**	47.3
Organizing Cabin.	43.7	**46.2**	Using ATM	**64.9**	50.6
Packing	**39.1**	38.3			
mAP				39.9	**41.7**

Table 4: Comparison of the per-class breakdown AP on the work subset of ActivityNet.