(Supplementary Material)
CAG-QIL: Context-Aware Actionness Grouping via Q Imitation Learning for Online Temporal Action Localization

Hyolim Kang, Kyungmin Kim, Yumin Ko, Seon Joo Kim

Yonsei University

{hyolimkang,kyungminkim,yuminko,seonjookim}@yonsei.ac.kr

1. Practical Usage of On-TAL

In this section, we briefly introduce several real-world applications of Online Temporal Action Localization (On-TAL).

1.1. Autonomous Driving

Consider the case of detecting a pedestrian in autonomous driving. Car must stop when it detects crossing pedestrian, maintain the status during the crossing, and move off at the endpoint of the detected action. In this case, detecting the whole action instance is exceptionally important because fragmented detection of the action instance can lead to an accident. Note that naive OAD extension is prone to fragmented detection as we mentioned in our paper, and ODAS only can detect start point of the action.

1.2. Robot Interaction

The situation above can be generalized to any computer vision task that involves alternating (dialogue-like) interaction. For example, a robot that actively interacts with the environment has to *a*) distinguish action instances from irrelevant background frames and *b*) point out the exact time of the start and end of the action in the online manner, since it should behave *differently* when the action is in progress and when it is finished.

1.3. Event-Driven Surveillance Camera

On-TAL can do a significant role in *extremely long* streaming video processing. Assume a dashboard camera (or standalone security cam) with limited memory. Saving all the streaming input is infeasible because of memory constraint, so there is a clear need for recording salient part. When we use ODAS to deal with this, it can decide when to start recording, but it's endpoint is vague. On-TAL, on the other hand, can solve the problem more precisely because it can adaptively decide when to start and stop recording depending on what it is seeing.

1.4. Streaming Sports Video Understanding

When we conduct video understanding on online streaming baseball video, its main task would be to follow up the high level context of the game, like counting the number of strikes, or homeruns. Note that the consequence (one-base hit, homerun etc.) of the pitching is revealed at the very end of the action. Therefore, ODAS cannot solve this task because simply detecting the start point of pitching does not really tell us about what it resulted in at the end. Besides, dynamically generated "homerun clip" can be instantly used as *highlight replay* or *bookmark* point. This situation can be generalized to any on-air sports broadcasting.

2. Implementation Details

2.1. Model Architecture

OAD models (M_1, M_2 in the notation from our paper) consist of one-layer LSTM with 1024 hidden unit and two FC layers for classification head. It takes a feature sequence as an input and generate actionness score α (M_1) or class probability p (M_2). They are both trained with cross-entropy loss.

Context-aware agent Υ is composed of 3 embedding modules, two FC layers with 64 hidden units and LeakyReLU activation function. To learn actionness and class probability embedding, we used WeightedEmbedding module, while Embedding module from torch.nn is used for decision embedding.

For training, we set queue (q_a, q_d) length to 24 ($n = 24$) and exploit 80,000 expert transitions in THUMOS14 and 240,000 in Activitynet1.3. Replay memory size is

set to 200,000 when we apply the DQN algorithm [7]. AdamW [6] optimizer with $lr = 0.0001$ is used for CAG-QIL training, while $lr = 0.001$ is used for training OAD models.

Code snippet of WeightedEmbedding module is provided below.

```
class WeightedEmbedding(nn.Module):
    def __init__(self, vocab_size,
        dim):
        super().__init__()
        self.weight =
        nn.Parameter(
            torch.randn(
                vocab_size, dim
            )
        )
        self.to(DEVICE)

    def forward(self, x):
        # IN: x [B, L, C]
        # C == vocab_size
        # OUT: [B, L, E]
        # E == dim
        x = x.unsqueeze(3)
        # [B, L, C, 1]
        x = x*self.weight
        # [B, L, C, E]
        x = torch.sum(x, dim=2)
        # [B, L, E]
        return x
```

2.2. Training Algorithm

Algorithm 1 briefly describes how CAG-QIL is actually implemented. Most of the notations are shared with the main paper, except for some newly introduced functions for clarity. For instance, $q.update(x)$ function refers to two consecutive operations $q.dequeue(); q.enqueue(x)$ and $\epsilon\text{-}Greedy(d_{max})$ returns a random decision d_{random} with probability ϵ and returns d_{max} otherwise. Remember that

$$\delta^2(D, r) \triangleq \frac{1}{|D|} \sum_{(s_t, d_t, s_{t+1}) \in D} (Q_\theta(s_t, d_t) - (r + \gamma \max_{d_{t+1}} Q_\theta(s_{t+1}, d_{t+1}))^2,$$

as we discussed in the main paper.

When it comes to OpenAI gym framework, $env.step(action)$ is equivalent to the line 23, 14, 15, 16 and 17 in the algorithm, since it receives a decision and returns an updated state.

To select best-performing model, we utilized Hungarian F1 score. It employs Hungarian algorithm [5] to find the most appropriate one-to-one matching between the prediction and the ground truth proposals in a class-agnostic manner and calculates F1 score from the matching. We consistently tracked its moving average during the training procedure, and stored the model if the score exceeds past best-performing score. After the training, we evaluated $F1 > 0.5$ models in the whole train set and selected one with the highest *recall*.

Algorithm 1: CAG-QIL

Input: Expert Database E_{expert},
Precalculated α, p sequences $AP_{i=1}^{N}$
Output: Trained $Q_\theta(s, a)$

1 Initialize agent replay memory $M_{agent} \leftarrow \phi$
2 Initialize expert replay memory $M_{expert} \leftarrow E_{expert}$
3 **for** $episode = 1, 2, 3, 4...$ **do**
4 Get a random α, p sequence $AP \in AP_{i=1}^{N}$
5 Initialize q_a and q_d
6 Get α_1 from AP
7 Get p_1 from AP
8 $q_a.update(\alpha_t)$
9 $s_1 \leftarrow [q_a; q_d; p_1]$
10 $d_{max} \leftarrow max_d Q_\theta(s_1, d)$
11 $d_1 \leftarrow \epsilon\text{-}Greedy(d_{max})$
12 $q_d.update(d_t)$
13 **for** $t \leftarrow 2$ **to** $len(A)$ **do**
14 Get α_t from AP
15 Get p_t from AP
16 $q_a.update(\alpha_t)$
17 $s_t \leftarrow [q_a; q_d; p_t]$
18 Store a transition (s_{t-1}, d_{t-1}, s_t) in M_{agent}
19 Sample a minibatch D_{agent} from M_{agent}
20 Sample a minibatch D_{expert} from M_{expert}
21 $\theta \leftarrow \theta - \eta \nabla_\theta (\delta^2(D_{expert}, +0.1) + \delta^2(D_{agent}, -0.1))$
22 $d_{max} \leftarrow max_d Q_\theta(s_t, d)$
23 $d_t \leftarrow \epsilon\text{-}Greedy(d_{max})$
24 $q_d.update(d_t)$
25 **end**
26 **end**

2.3. Queue Initialization

In this section, we describe how to initialize q_a and q_d at the first timestep. We manually compute d_1 ($d_1 = 1$ if $\alpha_1 > 0.5$ $d_1 = 0$ otherwise) and fill initial q_d with it. q_a is also filled with the α_1, except the last index. Fig. 1 is provided for clear explanation.

2.4. Video Segments Used during Training

Due to varying characteristics of the datasets, there is a slight difference in the training method for CAG-QIL. To be specific, partial video segments are used in THUMOS14 for training, while whole videos are used in Activitynet1.3.

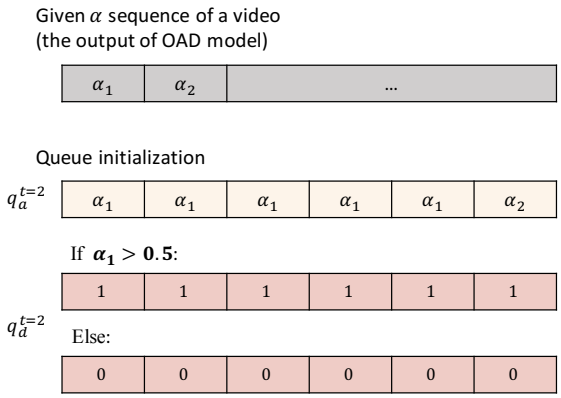

Figure 1. Illustration of queue initialization.

For example, suppose that we have a two-minute video. In THUMOS14, the 00:24 - 01:32 segment (the starting point and the length can vary) of the video can be used in the training procedure, while the whole 2 minute video is used for Activitynet1.3. This is because a video in THUMOS14 contains multiple actions but the number of videos is limited (only 200 training videos), whereas an Activitynet1.3 video contains only a small number of actions but the number of videos is relatively large.

Method	0.3	0.4	0.5	0.6	0.7
OAD-Grouping	33.3	28.0	22.0	16.8	10.4
CAG-GAIL	42.6	35.1	27.2	20.3	11.8
CAG-QIL	**44.7**	**37.6**	**29.8**	**21.9**	**14.5**

Table 1. Performance of CAG-GAIL, OAD-Grouping and CAG-QIL on THUMOS14.

3. Other Imitation Learning Algorithm

Generative Adversarial Imitation Learning (GAIL) [3] utilizes a discriminator network to distinguish where the state-action pair comes from and uses the network's output as the reward for the given state-action pair. The discriminator network is trained in the adversarial manner, as in [2]. In this setting, the reward will be high if the given state-action pair is more "expert-like", and low otherwise.

However, due to the inherent instability of the generative adversarial model, it suffers from extreme brittleness. Furthermore, since GAIL has a separate network to approximate the reward function, it has to harmonize multiple networks including the actor, the critic, and the reward networks, exacerbating the instability problem.

We tried to apply GAIL to our algorithm (namely CAG-GAIL) but it resulted in unsatisfactory performance compared to CAG-QIL. Experimental results is shown in the Table 1.

4. Evidence of CAG-QIL's Effectiveness

Figure 2 demonstrates action instance length distribution of each model. Note that CAG-RL does **NOT** alleviate the fragmented detection. Rather, it deteriorated the fragmentation issue, showing excessive number of very short proposals. Only CAG-QIL can generate action instances with reasonable length.

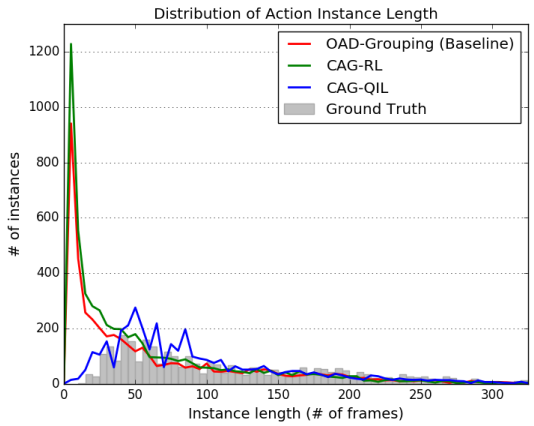

Figure 2. **Distribution of action instance length in THUMOS14.** Red, green and blue line represents distribution of action instance length generated by OAD-Grouping, CAG-RL, and CAG-QIL, respectively, while a bar graph represents ground truth distribution. QIL-trained model significantly reduces the number of short instances, matching distribution of ground truth.

5. Result on Baseball Database (BBDB)

In addition to THUMOS [4] and ActivityNet [1], we apply On-TAL to Baseball Database (BBDB), which was introduced in [8]. BBDB is a baseball dataset for multiple video understanding tasks, *e.g.* action recognition, temporal action localization, *etc.* It provides temporal boundary annotations of 30 action classes collected in a semi-automatic way.

Real-time sports broadcasting system is one of the most practical applications of On-TAL, in which other existing tasks such as TAL and OAD cannot be utilized. In the accompanying video, we present a demonstration of applying our framework for online baseball play-by-play system.

Here, we show the effectiveness of our proposed algorithm, CAG-QIL, on BBDB. Table 2 presents the action detection results of OAD-Grouping and CAG-QIL. CAG-QIL especially shows better performance than OAD-Grouping in long actions, like *Homerun*.

6. How about extending ODAS?

Our proposed method, CAG-QIL, is built on methods for OAD and showed outstanding results in Online Temporal Action Localization (On-TAL) and Online Detection

Dataset	Class	Length	OAD-Grouping	CAG-QIL
BBDB	Ball	5.0	38.0	34.1
	Base on balls	5.0	1.1	0.7
	Strike	5.0	37.5	33.5
	Swing and a miss	5.0	33.5	27.6
	Strike out	5.0	5.7	4.3
	Foul	7.0	39.0	48.0
	Hit by pitch	7.0	23.0	18.3
	Wild pitch	9.0	7.8	19.2
	Passed ball	9.0	0.0	0.0
	Bunt out	11.0	14.7	16.7
	Sacrifice bunt out	11.0	37.0	44.8
	Error	11.0	0.5	2.3
	Infield hit	11.0	3.9	12.2
	Bunt hit	11.0	24.1	24.5
	Pickoff out	11.0	0.0	0.0
	Fly out	11.0	9.5	29.9
	Foul fly out	11.0	33.4	39.1
	Ground out	11.0	2.9	12.1
	Stealing base	11.0	18.1	13.8
	Touch out	11.0	7.7	19.5
	Line-drive out	11.0	9.3	21.3
	Tag out	11.0	2.1	5.7
	Caught stealing	11.0	14.2	18.1
	One-base hit	12.0	17.0	38.3
	Double play	13.0	8.5	15.6
	Two-base hit	14.0	27.9	41.9
	Bunt foul	14.0	5.1	8.6
	Home in	15.0	3.0	7.3
	Three-base hit	16.0	3.5	5.0
	Homerun	16.0	14.5	34.0
	Average		14.8	19.8

Table 2. BBDB Class AP at tIoU 0.7.

of Action Start (ODAS). One might think extending ODAS to ODAE (Online Detection of Action End) and matching the start-end point also can solve On-TAL. However, the start-end matching is not trivial in that there's no guarantee of alternating start-end point detection. For example, the detected start/end point sequence can be "start-start-end-start-end-end..." unless we use regularization for alternating constraint. Therefore, designing appropriate regularization method for alternating start-end detection or finding algorithm for proper start-end point matching can be another good direction for solving On-TAL. We leave this for future work.

References

[1] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015. 3

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014. 3

[3] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*. 2016. 3

[4] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. http://crcv.ucf.edu/THUMOS14/, 2014. 3

[5] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 1955. 2

[6] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 2

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015. 2

[8] Minho Shim, Young Hwi Kim, Kyungmin Kim, and Seon Joo Kim. Teaching machines to understand baseball games: Large-scale baseball video database for multiple video understanding tasks. In *Proceedings of the European Conference on Computer Vision*, pages 404–420, 2018. 3