Supplementary Material: Progressive Relation Learning for Group Activity Recognition

Guyue Hu[1,3*] Bo Cui[1,3] Yuan He[1,3] Shan Yu[1,2,3]

In this supplementary material, **(1)** we firstly introduce some details about the standard flowchart of the A3C algorithm [3]. **(2)** We then take the proposed FD agent as an example to show how to specify the state, action and reward in the standard flowchart A3C algorithm. **(3)** In addition, we provide additional result and analysis about the confusion matrix on CAD dataset [1].

1. The details of A3C algorithm [3]

The asynchronous advantage actor-critic (A3C) algorithm [3] is a classical optimization algorithm for reinforcement learning, which contains many asynchronous updating Actor-Critic agents [2]. In A3C algorithm, there are numerous agent copies (workers) and each of them runs in an independent CPU thread/core. Each agent copy (worker) accumulates gradients to update a set of the globally shared parameters (including the value function parameters θ_v and the policy parameters θ). The global updating occurs when any worker runs every τ_{max} (updating interval) steps or reaches a terminal state. Therefore, the running process of every worker (CPU thread/core) is relatively independent, and the globally shared parameters are updated asynchronously by these workers.

Specifically, the agent copy (Actor-Critic agent) in each thread maintains both a thread-specific policy $\pi(A_\tau|S_\tau;\theta')$ (also named *Actor*) to generate actions and a thread-specific estimation of value function $V(S_\tau;\theta'_v)$ (also named *Critic*) to assess values for corresponding states. The accumulated reward at the step τ is $R_\tau = \sum_{i=0}^{k-1}\gamma^i r_{\tau+i} + \gamma^k V(S_{\tau+k};\theta'_v)$, where γ is the discount factor, r_τ is the reward at the τth step, and k varies from 0 to τ_{max}. In addition, the objective functions of A3C algorithm are formed from the advantage function [3] and entropy regularization term [4]. The advantage function can be calculated by $R_\tau - V(S_\tau;\theta'_v)$, and the entropy of policy π is denoted as $H(\pi(S_\tau;\theta'))$. Once a global updating is triggered, the gradients of *Critic* are accumulated via Eq. S1 to update the **globally shared value function parameters** θ_v. Similarly, the gradients of *Actor* are accumulated via Eq. S2 to update the **globally shared policy parameters** θ.

$$d\theta_v \leftarrow d\theta_v + \nabla_{\theta'_v}\left(R_\tau - V(S_\tau;\theta'_v)\right)^2/2, \tag{S1}$$

$$\begin{aligned} d\theta \leftarrow d\theta &+ \nabla_{\theta'} log\pi(A_\tau|S_\tau;\theta')\left(R_\tau - V(S_\tau;\theta'_v)\right) \\ &+ \beta\nabla_{\theta'}H(\pi(S_\tau;\theta')) \end{aligned} \tag{S2}$$

where β controls the strength of the entropy regularization term. After global updating at this time, the local agent synchronizes its parameter values from the updated globally shared parameters, *i.e.*

$$\theta' = \theta, \theta'_v = \theta_v. \tag{S3}$$

Finally, the local agent clears its gradient caches and starts to accumulate new gradients again. Because there are numerous local agent copies in many independent CPU threads, the trigger times of global updating are completely determined by the running speed of each thread.

The most important contribution of A3C algorithm is introducing the *Asynchronous Updating* policy. The policy involves numerous independent workers, whose implementation relies on the numerous threads of multi-core CPU. As stated by [3], although it works on a multi-core CPU, it achieves better results, in far less time than previous GPU-based algorithms, using far less resource than massively distributed approaches.

2. Training flowchart for the FD agent

The proposed feature-distilling (FD) agent and relation-gating (RG) agent are both optimized by the A3C algorithm [3]. The standard flowchart of A3C algorithm is introduced above, and we can obtain the detailed pipeline of agent training stage by simply specifying the state, action, and reward defined for each agent in the main text. The detailed pipeline of FD agent is shown in the Algorithm S1, only one thread-specific agent is shown to illustrate a minimal flowchart. In practice, we apply 16 threads (workers) to asynchronously update the globally shared parameters, and we stop all the threads once any thread have run 2 hours ($MaxRuntime$). The stop command is controlled by a globally shared variable, $Runtime$, which records the longest runtime of all the threads. The runtimes of threads are obtained by the public Python module *time* (https://docs.python.org/3/library/time.html).

Algorithm S1: Flowchart for each learning thread of the FD agent in A3C algorithm

// Assume globally shared variable $Runtime$ denotes runtime of threads:
// Assume globally shared parameter vectors θ and θ_v respectively for policy and value function of the FD agent
// Assume corresponding thread-specific parameter vectors θ' and θ'_v
// Terminal action: all the elements of action are "keep distilled"
Input: frozen semantic relation graph network *SRG*, frozen relation-gating agent *RG_agent*
Data: Trainning set $\mathcal{D}$ of group activity videos
start_time $= time.time()$ *// Mark start time of the thread*
repeat
  $I, l \leftarrow random\_choice(\mathcal{D})$ *// Randomly choose one video sample I and corresponding label l*
  $f_{whole} = Feature\_loader(I)$ *// Load low-level individual features*
  $A_0(M_0)$ = *random_action_mask(F_d) // F_d "True" (distilled) are randomly placed in the binary action mask*
  Perform random action $f_{distill} \leftarrow Feature\_distilling(f_{whole}, A_0)$
  Get intial state $S_1 = \{f_{whole}, f_{distill}, A_0\}$
  Get intial probability $p_1 = softmax(RG\_agent(SRG(f_{distill})))$
  Initialize thread step counter $\tau \leftarrow 1$
  repeat
    Reset global gradients: $d\theta \leftarrow 0$, $d\theta_v \leftarrow 0$
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $\tau_{start} = \tau$
    repeat
      Get action A_τ (mask M_τ) according to the policy output $\pi(A_\tau|S_\tau;\theta')$
      Perform action $f_{distill} \leftarrow Feature\_distilling(f_{whole}, A_\tau)$
      Get new state $S_{\tau+1} = \{f_{whole}, f_{distill}, M_\tau\}$
      Get new probability $p_{\tau+1} = softmax(RG\_agent(SRG(f_{distill})))$
      $r_\tau = reward(p_\tau, p_{\tau+1}, l)$ // Eq.13 *in the main text*
      $\tau = \tau + 1$
    until $\tau - \tau_{start} == \tau_{max}$ *or* $A_{\tau-1}$ *is terminal action*;
    $R = \begin{cases} 0 & \text{if } A_{\tau-1} \text{ is terminal action} \\ V(S_\tau;\theta'_v) & \text{for other actions} \end{cases}$
    for $i \in \{\tau-1, ..., \tau_{start}\}$ **do**
      $R \leftarrow r_i + \gamma R$
      Accumulate gradients w.r.t θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} log\pi(A_i|S_i;\theta')(R - V(S_i;\theta'_v)) + \beta\nabla_{\theta'} H(\pi(S_i;\theta'))$ // Eq. S2
      Accumulate gradients w.r.t θ'_v: $d\theta_v \leftarrow d\theta_v + \nabla_{\theta'_v}(R - V(S_i;\theta'_v))^2/2$ // Eq. S1
    end
    Perform asynchronous update of θ with $d\theta$ and θ_v with $d\theta_v$
  until $\tau \geq T_{max}$ *or* $A_{\tau-1}$ *is terminal action*;
  if *Runtime* $<$ *MaxRuntime* **then** *// Ensure that the shared Runtime haven't reached MaxRuntime in other threads*
    Runtime $= time.time() -$ *start_time* *// Calculate runtime of the thread*
  end
until *Runtime* $\geq$ *MaxRuntime*; *// The max runtime (MaxRuntime) is set as 2 hours in our paper.*
return θ, θ_v *// Globally shared parameters of the FD agent*

3. Additional Confusion Matrix

To better understand the performance of the proposed framework, the additional confusion matrix of our PRL on the CAD dataset [1] is shown Fig. S1. The results clearly show that our PRL has nearly solved the recognition of activities "Queuening" and "Talking", which proves the effectiveness of the proposed framework. However, the class "Waiting" is seriously confused by the class "Moving" probably because some "Moving" activities under the scene of crossing street are ill co-occur with some "Waiting" activities in the CAD dataset. Data cleaning or more training data are required to distinguish these two categories.

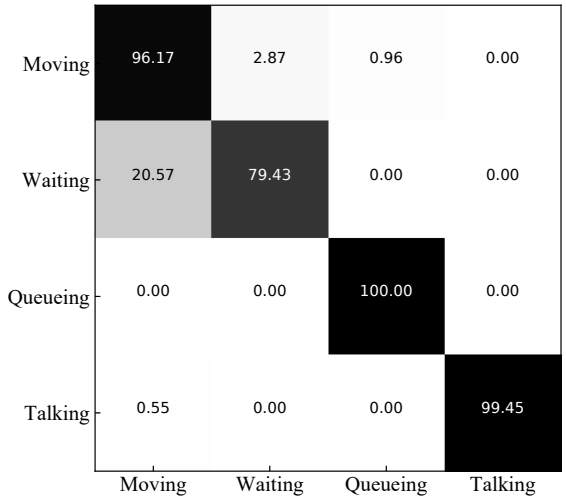

Figure S1: Confusion matrix on the CAD dataset.

References

[1] Wongun Choi, Khuram Shahid, and Silvio Savarese. What are they doing?: Collective activity classification using spatio-temporal relationship among people. In *ICCV Workshops*, pages 1282–1289. IEEE, 2009.

[2] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *NIPS*, pages 1008–1014, 1999.

[3] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[4] Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.