G-TAD: Sub-Graph Localization for Temporal Action Detection – Supplementary Material –

https://www.deepgcns.org/app/g-tad

Mengmeng Xu, Chen Zhao, David S. Rojas, Ali Thabet, Bernard Ghanem
Visual Computing Center
King Abdullah University of Science and Technology (KAUST), Saudi Arabia
{Mengmeng.Xu, chen.zhao, davidsantiago.blanco, ali.thabet, bernard.ghanem}@kaust.edu.sa

1. Derivation and Efficient Implementation of Eq. 4

In this section, we provide the derivation of Eq. 4 in the paper (listed here in the following). We also show that Eq. 4 can be efficiently implemented by zero-padded 1D/edge convolutions.

$$\mathcal{H}(X, \mathcal{E}, W) = ReLU[W_\alpha X + W_\beta X A_t^f + W_\gamma X A_t^b +_\phi X A_s + X]. \tag{4}$$

1.1. Derivation of Eq. 4

a) Temporal Graph Convolution. We first provide the derivation for temporal graph convolution.

The temporal forward edges $\mathcal{E}_t^f$ and backward edges $\mathcal{E}_t^b$ are formulated as

$$\mathcal{E}_t^f = \{(v_i, v_{i+1}) |\ i \in \{1, 2, \dots, L-1\}\}, \mathcal{E}_t^b = \{(v_i, v_{i-1}) |\ i \in \{2, \dots, L-1, L\}, \tag{1}$$

The corresponding adjacency matrices A_t^f, A_t^b can be present by L vectors, respectively, shown in Eq. 3. We use $e_k \in \mathbb{R}^L$ to present the vector in which the k-th element is one but the others are zeros.

$$A_t^f = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & & 0 & 0 \\ 0 & 1 & 0 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix} = [e_2, e_3, \dots, e_L, 0]$$

$$A_t^b = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ 0 & 0 & 0 & & 0 & 0 \\ \vdots & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix} = [0, e_1, e_2, \dots, e_{L-1}],$$

$$\Rightarrow A_t^f = [A_t^b]^T \tag{2}$$

Given the input $X \in \mathbb{R}^{C \times L}$, after temporal graph convolution, the output, X_t, becomes

$$\begin{aligned} X_t &= f_{agg}(X, A_t^f, W_f) + f_{agg}(X, A_t^b, W_b) = W_{f,0} X + W_{f,1} X A_t^f + W_{b,0} X + W_{b,1} X A_t^b \\ &= (W_{f,0} + W_{b,0}) X + W_{f,1} X A_t^f + W_{b,1} X A_t^b. \end{aligned} \tag{3}$$

Here W_* is the trainable weights in the neural network.

b) Semantic Graph Convolution. It is straightforward to obtain Eq. 4 for semantic graph convolution.

1.2. Efficient Implementation of Eq. 4

In implementation of Eq. 4, we use an efficient zero-padded 1D convolution and edge convolution for temporal graph convolution and semantic graph convolution, respectively. In the following, we provide proof that our efficient implementation is equivalent to Eq. 4.

a) Temporal Graph Convolution.

If a 1D convolution has kernel size 3, the weight matrix is a 3D tensor in $\mathbb{R}^{3\times C\times C}$. We denote the matrix as $W_{conv1} = [W_1, W_2, W_3], W_{1,2,3} \in \mathbb{R}^{C\times C}$. Given the same input $X = [x_1, x_2, ..., x_L]$, we pad zero on the input, $x_0 = x_{L+1} = 0 \in \mathbb{R}^C$. The output of 1D convolution can be written as $Y = [y_1, y_2, ..., y_L] \in \mathbb{R}^{C\times L}$

$$y_k = [W_1 x_{k-1} + W_2 x_k + W_3 x_{k+1}], k = 1, 2, \ldots, L \tag{4}$$

We can prove that $X_t = Y$ by multiplying e_k on both sides in Eq. 5. Please be noted that W_* is the trainable weights in the neural network. We can assume $W_1 = W_{b,1}, W_3 = W_{f,1}, W_2 = W_{f,0} + W_{b,0}$

$$\begin{aligned} X_t e_k &= W_{f,0} X e_k + W_{f,1} X A_t^f e_k + W_{b,0} X e_k + W_{b,1} X A_t^b e_k \\ &= (W_{f,0} + W_{b,0}) x_k + W_{f,1} X [0, e_1, e_2, \ldots, e_L]^T e_k + W_{b,1} X [e_2, e_3, \ldots, e_L, 0]^T e_k \\ &= (W_{f,0} + W_{b,0}) x_k + W_{f,1} X e_{k+1} + W_{b,1} X e_{k-1} \\ &= (W_{f,0} + W_{b,0}) x_k + W_{f,1} x_{k+1} + W_{b,1} x_{k-1} \\ &= W_1 x_{k-1} + W_2 x_k + W_3 X_{k+1} \end{aligned} \tag{5}$$

b) Semantic Graph Convolution. In the semantic graph, edge convolution is directly used, so proof is done.

2. Training Details

Semantic Edges from Multiple Levels. In G-TAD, we use multiple GCNeXt blocks to adaptively incorporate multi-level semantic context into video features. After that, SGAlign layer embeds each sub-graph by concatenating aligned features from temporal and semantic graphs. However, it is not necessary to consider **only the last** GCNeXt semantic graphs to align the semantic feature. Last row in Tab. 1 present one more experiment that takes **the union of** semantic edges from all GCNeXt blocks to aggregate the semantic feature. We can find that the semantic context also helps to improve model performance under this setup.

Table 1. **Ablating SGAlign Components.** We disable the sample-rescale process and the feature concatenation from the semnantic graph for detection on ActivityNet-1.3. The rescaling strategy leads to slight improvement, while the main gain arises from the use of context information (semantic graph).

SGAlign		tIoU on Validation Set			
Samp.	Concat.	0.5	0.75	0.95	Avg.
✗	✗	49.84	34.58	8.17	33.78
✓	✗	49.86	34.60	**9.56**	33.89
✓	✓	**50.36**	34.60	9.02	**34.09**
✓	all	50.26	**34.70**	8.52	33.95

2D Conv. for Sub-Graph Localization. Once we get the *sub-graph feature* from SGAlign layer, instead of using three fully connected (FC) layers regress to g_c, we can arrange the anchors in a 2D $L \times L$ map based on the start/end time, and set zeros to the map where is no pre-designed anchors (e.g. $t_s > t_e$). In doing so, we can use 2D CNNs to regress to a g_c map that arranged by the same order. We call the predicted matrix **IoU map**.

The neighbouring anchors in the 2D IoU map have similar boundary locations. Thus we can use the proposal-proposal relationship in the 2D convolutions. We set kernel size to 1, 3, and 5, and the results are shown in Tab. 2. We do not observe any significant benefit from 2D convolutions.

Table 2. **The model performance when we use 3 2D convolution layers to predict IoU map.**We set kernel size to 1, 3, and 5, and collect result on ActivityNet1.3. We do not observe any significant benefit from 2D convolutions.

Conv. on IoU map		mAP on Validation Set			
Kernel Size	Padding	0.5	0.75	0.95	Avg.
(1,1)	(0,0)	**50.25**	34.66	**9.29**	34.08
(3,3)	(1,1)	**50.25**	**34.94**	7.74	**34.10**
(5,5)	(2,2)	49.88	34.39	8.96	33.77