Supplementary Material for: Adaptive Interaction Modeling via Graph Operations Search

Haoxin Li[1], Wei-Shi Zheng[2,3,5,*], Yu Tao[2,4], Haifeng Hu[1,*], Jian-Huang Lai[2]
[1]School of Electronics and Information Technology, Sun Yat-sen University, China
[2]School of Data and Computer Science, Sun Yat-sen University, China
[3]Peng Cheng Laboratory, Shenzhen 518005, China
[4]Accuvision Technology Co. Ltd.
[5]Key Laboratory of Machine Intelligence and Advanced Computing, Ministry of Education, China
lihaoxin05@gmail.com, wszheng@ieee.org, gytaoyu@hotmail.com
huhaif@mail.sysu.edu.cn, stsljh@mail.sysu.edu.cn

1. Additional Results

1.1. Results on Test Set

We report the results on validation set in our paper for comprehensive comparison, because most of other methods only report validation results due to the withheld test labels. In Table 1, we compare the results on test set, which show that our method also achieves competitive performances with other available methods.

Methods	V1 test acc	V2 test acc
NonLocalI3D+GCN	45.0	-
CPNet	-	57.6
TSM	46.1	59.9
ECO	42.3	-
S3D	42.0	-
GST	-	61.2
STM	43.1	63.5
Ours	46.3	62.6

Table 1. Top-1 accuracy (%) of RGB based methods on test set.

1.2. Model Size and Inference Time

In Table 2, we evaluate the number of parameters and MACs (multiply–accumulate operations) of our model using a public available tool (`https://github.com/sovrasov/flops-counter.pytorch`). Our method would not increase the model size too much (comparable with NonLocalI3D), but still boost performance.

In terms of inference time, our framework takes around 0.12 seconds per video with 32 sampled frames on a single GTX 1080TI GPU, which is still towards real time and would not lead to excessive latency.

*Corresponding author

Model	Params	GMACs
NonLocalI3D	35.3 M[†]	167.5[†]
Our backbone	32.1 M	127.6
Our whole model	37.3 M	138.4

Table 2. Number of parameters and MACs ([†] Calculated in [7]).

2. Implementation Details

Backbone. We use I3D-ResNet [9, 10], which inflates 2D convolution kernels into 3D kernels for initialization [2], as our backbone (Table 3) to extract basic features. It is inflated from ResNet-50 [6] with ImageNet [4] pretrained parameters, and it extracts video features after "res5" with 2048 channels from 32 uniformly sampled frames. For computation efficiency in graph reasoning, we reduce the feature dimension from 2048 to 256 with a FC layer.

RPN. We use RPN [5] model with ResNet-50 and FPN to extract region proposals. The RPN model is pre-trained on the MSCOCO object detection dataset [8]. To match the output time dimension of the "res5" feature maps, we sample 16 frames from 32 input frames (1 frame every 2 frames) to extract region proposals. Top 10 class-agnostic object bounding boxes are extracted for each frame.

Graph operations settings. The shapes of all the transform matrices $\boldsymbol{W}_*$, $\boldsymbol{V}_*$ in the graph operations are set as $256 \times C_{in}$, where C_{in} differs from operation to operation. The shapes of the affinity weights $\boldsymbol{U}_*$ are set as $C_{in} \times C_{in}$. The affinity matrices are row-normalized to keep the sum of the affinities connected to each node to be 1. The size of temporal convolution kernel $\boldsymbol{W}_t$ is set to 7. We employ Layer Normalization [1] followed by LeakyReLU as the nonlinear activation function of each operation.

layer		output size
conv1	5×7×7, 64, stride 1,2,2	32×112×112
pool1	3×3×3, max, stride 1,2,2	32×56×56
res2	$\begin{bmatrix} 3\times1\times1, 64 \\ 1\times3\times3, 64 \\ 1\times1\times1, 256 \end{bmatrix} \times 3$	32×56×56
pool2	3×1×1, max, stride 2,1,1	16×56×56
res3	$\begin{bmatrix} 3\times1\times1, 128 \\ 1\times3\times3, 128 \\ 1\times1\times1, 512 \end{bmatrix} \times 4$	16×28×28
res4	$\begin{bmatrix} 3\times1\times1, 256 \\ 1\times3\times3, 256 \\ 1\times1\times1, 1024 \end{bmatrix} \times 6$	16×14×14
res5	$\begin{bmatrix} 3\times1\times1, 512 \\ 1\times3\times3, 512 \\ 1\times1\times1, 2048 \end{bmatrix} \times 3$	16×7×7

Table 3. Our backbone ResNet-50 I3D model. The kernel size and output size are represented as $T \times H \times W$. The input size is 32×224×224.

Training. We train our model in the following steps:

1. Train the backbone on the target datasets.
2. Fix the backbone. Train the weights in all the graph operations and the structure weights alternatively to learn adaptive structures until the adaptive structures are stable. The SGD optimizer is used for the weights in all the graph operations, and the Adam optimizer is used for the structure weights. The learning rate of the weights in all the graph operations is 0.01, and the learning rate of the structure weights is 0.0001.
3. Fix the structure weights. Train the weights in all the graph operations with discrete structures. SGD optimizer is used and the learning rate is set to 0.001. The learning rate is divided by 10 when the validation loss doesn't decline for 5 epochs. The training is stopped when the validation loss doesn't decline for 5 epochs with learning rate 0.0001.
4. Train the weights in all the graph operations and the backbone jointly. SGD optimizer is used and the learning rate is 0.0001.

Testing. On the testing stage, we sample 5 clips in each video and use the mean score for classification.

Data augmentation. We divide the video into 32 segments and randomly sample one frame in each segment, in order to obtain different samples from the same video for augmentation. We also randomly crop and horizontally flip all the sampled frames of the same video. It should be noted that some categories are relevant to directions, so that we do not apply horizontal flipping to these videos.

3. Analysis of Architecture Search Framework

3.1. Fixing Substructures

Table 4 compares the performance of learning with original graph operations and fixed substructures. It is observed that learning with fixed substructures obtains higher accuracy, perhaps because it simplifies the optimization with fewer structure weights and also implicitly deepens the structures. In addition, learning with fixed substructures converges faster than learning with original graph operations in our experiments, which reduces the searching time.

Settings	V1 Val Acc	V2 Val Acc
Ori Ops	51.0	63.1
Fixed Subs	51.4	63.5

Table 4. Interaction recognition accuracy (%) comparison of different search space. "Ori Ops" means original graph operations are used as basic operations in the search space, and "Fixed Subs" means the fixed substructures are used as basic operations to search the structures.

3.2. Diversity Regularization

To validate the effect of diversity regularization, we search non-adaptive structures with original graph operations on Something-Something-V1 dataset and compare the searched structures without and with diversity regularization (variance loss in Equation (13) in our paper) in Figure 1. It is observed that the structure learned without variance loss tends to only select "node attention", which hampers complex relation modeling and also obtains unsatisfactory performance (Table 5). In contrast, the structure learned with variance loss selects diverse graph operations, which enhances the ability of interaction modeling and gains better recognition results. Therefore, we all use diversity regularization in other experiments.

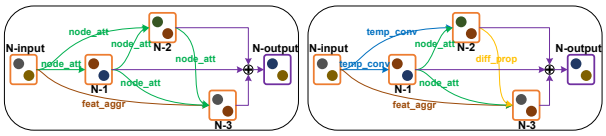
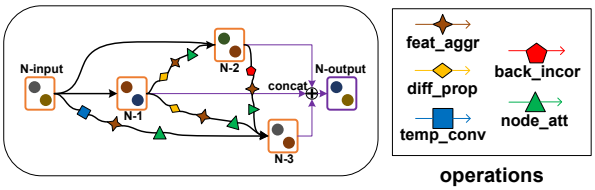


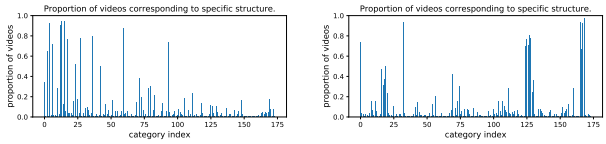
Figure 1. The learned structures without (left) and with (right) variance loss for diversity regularization.

Settings	V1 Val Acc
w/o var loss	49.9
with var loss	50.7

Table 5. Interaction recognition accuracy (%) comparison about diversity regularization.

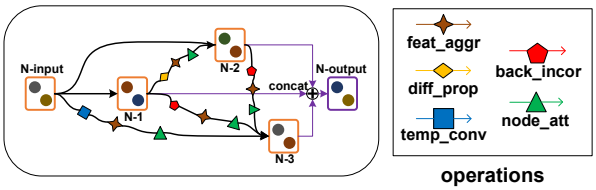


(a) Structure

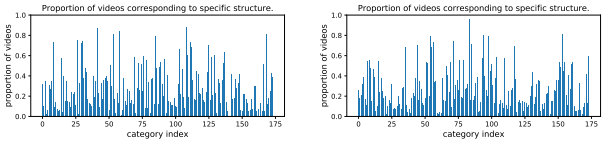


(b) Something-Something-V1 (c) Something-Something-V2

Figure 2. Example 1: proportion of videos per class corresponding to the structure (a) in the original dataset (b) and the transferred dataset (c).

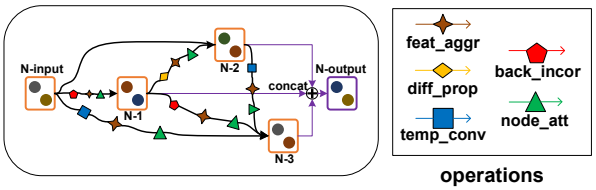


(a) Structure

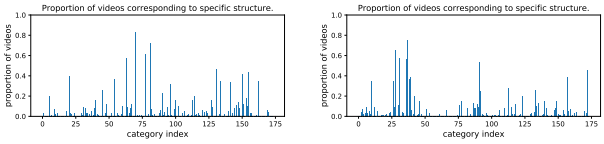


(b) Something-Something-V1 (c) Something-Something-V2

Figure 3. Example 2: proportion of videos per class corresponding to the structure (a) in the original dataset (b) and the transferred dataset (c).



(a) Structure



(b) Something-Something-V1 (c) Something-Something-V2

Figure 4. Example 3: proportion of videos per class corresponding to the structure (a) in the original dataset (b) and the transferred dataset (c).

3.3. Transferability of Adaptive Structures

In order to verify the transferability of the adaptive structures, we learn the adaptive structures on one dataset, and then fix the structure weights and train the rest learnable weights on the other dataset. The results are shown in Table 6. It is observed that the interaction recognition performance does not decline obviously (0.4% for Something-Something-V1 dataset and 0.1 % for Something-Something-V2 dataset), which indicates that the adaptive structures can transfer across datasets with minor performance degradation.

transfer settings	V2 → V1	V1 → V2
accuracy	51.0	63.4

Table 6. Interaction recognition accuracy (%) of transferring the adaptive structures across datasets. "dataset1 → dataset2" means the structure weights are trained in dataset1 and fixed in dataset2.

We further show the proportion of videos per class corresponding to some structures in the original and transferred datasets. Figure 2 to Figure 4 show three examples of the structure and its corresponding interaction category distributions in the original dataset (Something-Something-V1) and transferred dataset (Something-Something-V2). According to the category index and the label lists, we observed that the dominant interaction categories are semantically similar in the two datasets. In both the original dataset and the transferred dataset, the dominant categories are about camera motion, pushing/poking/throwing something, moving something away or closer in the three examples respectively. These results fully illustrate that the adaptive structures are learned according to some interaction characteristics, which can transfer across datasets and also help us understand the relations between the structures and the interactions.

3.4. Computation Cell

We compare the performance of the computation cells with different number of intermediate supernodes, and the results are shown in Table 7. It is observed that computation cells with 3 and 4 intermediate supernodes obtain better performances than that with 2 intermediate supernodes. But using 4 intermediate supernodes could not further improve the performance, perhaps because the structures with 4 intermediate supernodes are too complex, which leads to highly complex back-propagation and optimization difficulties. Therefore, we use 3 intermediate supernodes in our experiments.

3.5. Deeper Structures

We attempt to stack multiple searched computation cells to construct deeper structures. Table 8 compares the results with 1 and 2 stacked cells. It is observer that deeper struc-

Settings	V1 Val Acc	V2 Val Acc
2 supernodes	50.6	63.1
3 supernodes	51.4	63.5
4 supernodes	51.3	63.5

Table 7. Interaction recognition accuracy (%) comparison of different number of intermediate supernodes in the computation cell.

Settings	V1 Val Acc	V2 Val Acc
1 cells	51.4	63.5
2 cells	50.7	63.0

Table 8. Interaction recognition accuracy (%) comparison of stacking multiple computation cells.

tures with 2 stacked cells gain no improvements, perhaps due to the gap between search and evaluation [3] and overfitting. Perhaps, search with multiple stacked cells could boost the performance but it leads to heavy computation consumption. For simplicity, we use only 1 cell in our experiments.

4. Analysis of Graph Operations

4.1. More Successful and Failed Cases

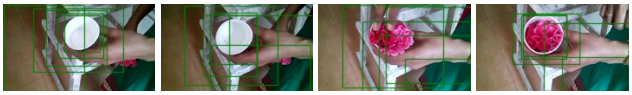
We show more successful and failed cases to indicate the effects of different operations in Figure 5.

The *feature aggregation* successfully recognizes the simple interaction in (a), but it fails in case (b), (c), and (d) because some detailed relations need to be modeled with other operaiotns.

In case (b), the *feature aggregation* easily recognizes the interaction as "Letting something roll down a slanted surface". The key to distinguish the two interactions is the differences between rolling and sliding. The *difference propagation* focuses on the differences between detected objects, which enables it to capture the changes of the sliding down object so that it successfully classifies the interaction as sliding.

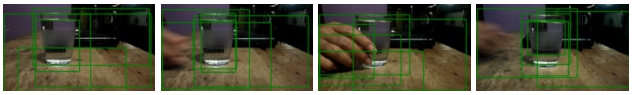
In case (c), the *feature aggregation* mistakenly recognizes the interaction as "Moving something and something closer to each other" since the two objects are indeed closing. However, the key to identify is whether the two objects are both moved. The *temporal convolution* aims to capture the evolution of the interaction and it could observe that one of the objects is always static, which makes the interaction distinguishable.

As for case (d), something is pushed but did not fall down but the *feature aggregation* misclassifies it as "Pushing something off of something". The background would change dramatically if the box falls off, so the *background incorporation* modeling the relations between the nodes and the background helps to identify the action. On the contrary, those operations relying on detected objects easily fail because the table is not detected by RPN model.



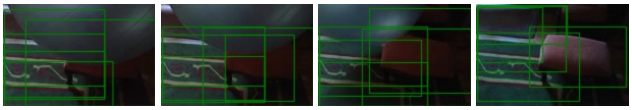
(a) Stuffing something into something

(b) Putting something that can't roll onto a slanted surface, so it slides down



(c) Pushing something so that it slightly moves

(d) Pushing something with something



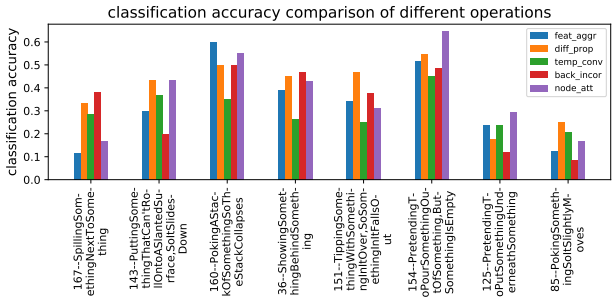
(e) Putting something on the edge of something so it is not supported and falls down

Figure 5. Successful and failed cases of different graph operations.

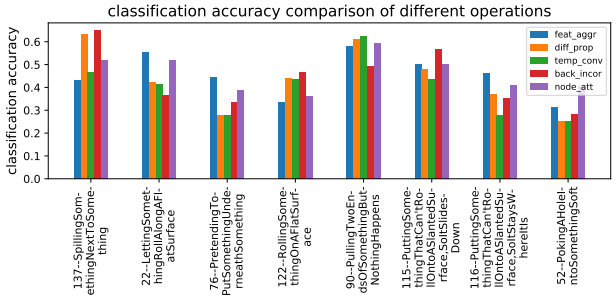
However, there are still many cases that are commonly misclassified by different graph operations and the searched structures, such as the example showed in Figure 5(e). The poor quality frames prevent any effective modeling based on RGB inputs. What's more, some confusing labels and incorrect detected object bounding boxes also hinder further improvements of interaction modeling, which need to be addressed in the future.

4.2. Accuracy of Each Graph Operation

To show the effects of different graph operations on different interaction categories, we compare the recognition accuracy of each graph operation on some interaction categories where different operations obtain quite different performances. The results are shown in Figure 6. It is observed that different operations perform differently on the same interaction category, since they tend to model different relations in videos. We can also observe that the *different propagation* and the *temporal convolution* generally work well on the interactions with detailed changes, such as spilling something, moving something slightly and rolling something, and the *background incorporation* would work well on some interactions with relations between different objects and the background, such as positional relations and relations to the surfaces.



(a) Something-Something-V1



(b) Something-Something-V2

Figure 6. Accuracy comparison of each graph operation on some interaction categories.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[3] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *arXiv preprint arXiv:1904.12760*, 2019.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[7] Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 7083–7093, 2019.

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *The European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[9] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7794–7803, 2018.

[10] Xiaolong Wang and Abhinav Gupta. Videos as space-time region graphs. In *The European Conference on Computer Vision (ECCV)*, pages 413–431, 2018.