Dynamic Spatio-Temporal Specialization Learning for Fine-Grained Action Recognition (Supplementary Material)

Tianjiao Li[1⋆], Lin Geng Foo[1]1, Qiuhong Ke[2], Hossein Rahmani[3], Anran Wang[4], Jinghua Wang[5], and Jun Liu[1⋆⋆]

[1] ISTD Pillar, Singapore University of Technology and Design
{tianjiao_li,lingeng_foo}@mymail.sutd.edu.sg, jun_liu@sutd.edu.sg
[2] Department of Data Science & AI, Monash University
qiuhong.ke@monash.edu
[3] School of Computing and Communications, Lancaster University
h.rahmani@lancaster.ac.uk
[4] ByteDance
anranwang1991@gmail.com
[5] School of Computer Science and Technology, Harbin Institute of Technology
wangjinghua@hit.edu.cn

1 Implementation Details of Improved SemHash

We follow the implementation details of the Improved Semhash as in [2, 3, 1], and list the detailed steps below for the sake of reproducibility. We omit the layer index and show the procedure for a specialized neuron n_i. For Improved Semhash, the inputs are gate parameters $\mathbf{g}_i \in \mathbb{R}^{N_{in}}$ and sampled noise $\boldsymbol{\epsilon} \in \mathbb{R}^{N_{in}}$, while the outputs are a binary vector $\mathbf{b} \in \{0,1\}^{N_{in}}$ for channel-wise architectural decisions, and a real vector $\mathbf{r} \in \mathbb{R}^{N_{in}}$ for the purposes of gradient computation.

During training, a noise vector $\boldsymbol{\epsilon} \in \mathbb{R}^{N_{in}}$ is first sampled from a Gaussian distribution with mean 0 and variance 1, and we add this noise to our gate parameters $\mathbf{g}_i$ to get $\mathbf{g}_{i,noisy} \in \mathbb{R}^{N_{in}}$ as follows:

$$\mathbf{g}_{i,noisy} = \mathbf{g}_i + \boldsymbol{\epsilon} \tag{1}$$

Then, we generate a real-valued vector $\mathbf{r} \in \mathbb{R}^{N_{in}}$ and a binary vector $\mathbf{b} \in \{0,1\}^{N_{in}}$:

$$\mathbf{r} = \sigma'(\mathbf{g}_{i,noisy}) \tag{2}$$

$$\mathbf{b} = \mathbb{1}(\mathbf{g}_{i,noisy} > 0) \tag{3}$$

where $\mathbb{1}$ is the indicator function, and σ' is the saturating sigmoid function [2, 3, 1]:

$$\sigma'(x) = max(0, min(1, 1.2\sigma(x) - 0.1)) \tag{4}$$

⋆ equal contribution
⋆⋆ corresponding author

with σ being the sigmoid function.

$\mathbf{b}$ and $\mathbf{r}$ are utilized in different ways. $\mathbf{b}$ is binary and is more suitable for the model during inference in Eq. 4 and Eq. 5 of the main paper. However, $\mathbf{b}$ is not suitable during training because most gradients $\frac{\partial \mathbf{b}}{\partial \mathbf{g}_i}$ are zero as a result of backpropagating through the indicator function. On the other hand, $\mathbf{r}$ is more suitable during training, and less suitable during inference, as the gradient $\frac{\partial \mathbf{r}}{\partial \mathbf{g}_i}$ is well-defined, but $\mathbf{r}$ is not binary.

During training, we randomly use $\mathbf{b}$ for half of the training samples and use $\mathbf{r}$ for the rest of the samples. When $\mathbf{b}$ is used, we follow [2, 3, 1] and define gradient $\frac{\partial \mathbf{b}}{\partial \mathbf{g}_i}$ to be $\frac{\partial \mathbf{r}}{\partial \mathbf{g}_i}$ in the backpropagation step.

At test time, we do not sample the noise and set $\boldsymbol{\epsilon} = \mathbf{0}$. We also only use the discretized $\mathbf{b}$ during forward propagation.

2 Upstream-Downstream Learning algorithm

An outline of our proposed Upstream-Downstream Learning algorithm is shown in Algorithm 1.

Algorithm 1 Upstream-Downstream Learning

Requires: Upstream parameters u (scoring kernels and gates) and Downstream parameters d (spatial and temporal operators)

for `epoch in max_epoch` **do**
  for `each iteration` **do**
    $\{X, Y\} \leftarrow D_{train}$
    Freeze Upstream Parameters u
    Update Downstream Parameters $\hat{d} \leftarrow d - \alpha \nabla_d \ell(u, d; X, Y)$
    $\{X_v, Y_v\} \leftarrow D_{val}$
    Freeze Downstream Parameters $\hat{d}$
    Update Upstream Parameters $u' \leftarrow u - \alpha \nabla_u \ell(\hat{u}, \hat{d}; X_v, Y_v)$
    Update Downstream Parameters $d' \leftarrow d - \alpha \nabla_d \ell(u', d; X, Y)$
    $u \leftarrow u'; d \leftarrow d'$
  end for
end for

3 Visualization of Spatial and Temporal Specializations

Ideally, different specialized neurons should optimize their architectures towards different extents of spatial or temporal specialization, to handle fine-grained differences over a large range of spatio-temporal variations. We visualize the proportion of channels that use the spatial or temporal operator for each specialized neuron in the three layers of our DSTS module in Fig. 1. In each layer, the specialized neurons are diverse in their specializations, and tend to use different proportions of the spatial and temporal operators. Some specialized neurons

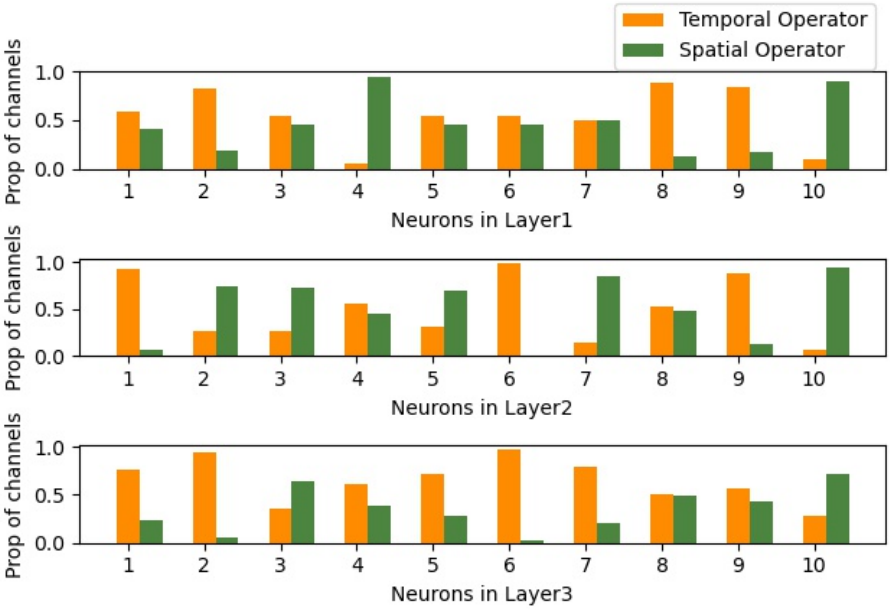

Fig. 1. Visualization of the proportion of channels that use the spatial or temporal operator for each specialized neuron. This visualization is taken from the DSTS module with a TPN backbone, trained on Diving48. Here, we visualize all 10 specialized neurons ($N = 10$) across 3 layers ($L = 3$). In each layer, we observe that there is a diversity of spatio-temporal specializations among the specialized neurons. For example, in layer 2, specialized neurons 7 and 10 use a lot more of the spatial operator and specialize in the spatial aspects, while specialized neurons 1, 6 and 9 use a lot more of the temporal operator and specialize in the temporal aspects. The others (specialized neurons 2, 3, 4, 5, 8) use both operators to a moderate extent.

focus more on the spatial aspect, some focus on the temporal aspect, while others are somewhat mixed. This shows that our spatio-temporal specialization provides our set of specialized neurons with diversified architectures and specializations, which collectively are capable of handling a large variety of spatial and temporal fine-grained differences.

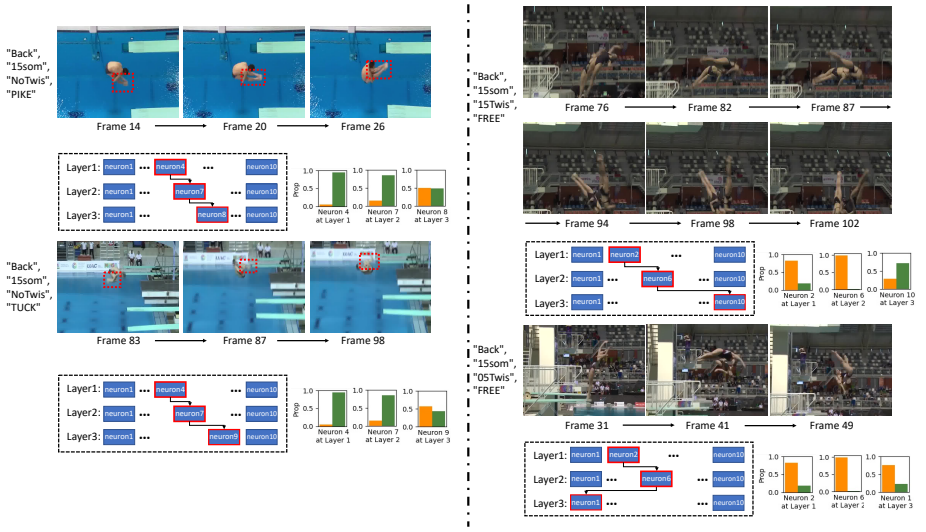

Fig. 2. Visualization of specialized neuron activation patterns by similar samples. The neurons depicted in this visualization have the same spatio-temporal specializations as those depicted in Fig. 1. (Left) Between these samples, observers have to distinguish between the shapes of the legs of the divers ("Pike" vs "Tuck") that have been indicated in red rectangles, which can be quite subtle. These similar samples activate the same specialized neurons in the first two layers (4 and 7 in layers 1 and 2 respectively), that correspond to the "spatial-specialized" neurons in Fig. 1. (Right) Between these samples, observers have to distinguish between the number of twists the divers do ("15Twis" vs "05Twis"), which can be discerned using the duration of the diver's twist. To visualize the duration aspect here, we show 6 key frames for the longer twist, and only 3 key frames for the shorter twist. These similar samples activate the same specialized neurons in the first two layers (2 and 6 in layers 1 and 2 respectively), that correspond to the "temporal-specialized" neurons in Fig. 1. Between both sets of samples, we can qualitatively observe the spatio-temporal specialization at work, where specialized neurons adjust their specializations to handle spatial and temporal fine-grained differences among similar subsets of samples.

4 Visualization of Activation of Neurons

We visualize the activation patterns of specialized neurons by similar samples in Fig. 2. We observe that similar samples tend to activate the same specialized neurons. For these similar samples, the same specialized neurons were activated in layer 1 and layer 2. This shows qualitatively that similar samples tend to activate the same specialized neurons via the synapse mechanism, which pushes the specialized neurons to learn to handle the fine-grained information relevant to these similar samples, instead of learning more common discriminative cues that are applicable to the more common samples. Moreover, we observe that samples on the left side tend to activate neurons that focus more on the spatial aspects, while the samples on the right side tend to activate the specialized neurons which focus more on the temporal aspects. This shows that our spatio-temporal specialization method is capable of dealing with spatial and temporal fine-grained differences by adaptively adjusting the proportion of spatial and temporal operators for each specialized neuron.

5 Skip Connection

We investigate whether the skip connection (as shown in Fig. 2 of the main paper illustrating our DSTS module) leads to better performance. Results are shown in Table 1. We remove the skip connection in the setting **DSTS w/o Skip Connection**, while keeping our design in **DSTS w/ Skip Connection**. We observe that adding the skip connection leads to slight improvement in performance. This shows that adding the output of the DSTS module (which are more specialized) to the output of the backbone (which are more general) leads to performance gains.

Table 1. Evaluation results (%) on the impact of the skip connection on Diving48.

Method	Top-1	Class-wise Acc
DSTS w/o Skip Connection	88.1	77.9
DSTS w/ Skip Connection	88.4	78.2

6 Scoring Kernel Shape

We evaluate the impact of using different shapes for the scoring kernel m, with results shown in Table 2. We design the scoring kernel to be of shape $N_{out} \times N_{in} \times 1 \times 1 \times 1$, for the best and most efficient performance.

Table 2. Evaluation results (%) for different shapes of scoring kernel m on Diving48.

Scoring Kernel Shape	Top-1	Class-wise Acc
$N_{out} \times N_{in} \times 1 \times 1 \times 1$	88.4	78.2
$N_{out} \times N_{in} \times 1 \times 3 \times 3$	88.3	78.0
$2N_{out} \times N_{in} \times 1 \times 1 \times 1$	88.2	77.7

7 $1 \times 1 \times 1$ Convolution

We evaluate the impact of using alternative shapes other than the $1 \times 1 \times 1$ convolution for processing the output Z to obtain Z' (as shown in the Fig. 3 of the main paper). Results are shown in Table 3. We choose the shape of $N_{out} \times N_{out} \times 1 \times 1 \times 1$ as it provides the best and most efficient performance.

Table 3. Evaluation results (%) for alternative shapes of the kernel used to process the output Z to obtain Z' on Diving48.

Kernel Shape	Top-1	Class-wise Acc
$N_{out} \times N_{out} \times 1 \times 1 \times 1$	88.4	78.2
$N_{out} \times N_{out} \times 1 \times 3 \times 3$	88.1	77.5
$N_{out} \times N_{out} \times 3 \times 1 \times 1$	88.0	77.8

8 Training Cost

We compare the overall training time of our method against the backbone on Diving48 dataset in Table 4. The experiments are conducted on Nvidia V100 GPU. We observe that, compared to the training of the backbone TPN, the overall training time increase brought by our method is not high, because our DSTS module is relatively small compared to the backbone TPN.

Table 4. Training cost and evaluation accuracy on Diving48 using TPN backbone. Note that performance using the backbone TPN is 86.2%, while applying our method onto those backbones achieves 88.4%, bringing performance gains of 2.2%.

Method	Training Time (hrs)	Acc. (%)
Backbone	9	86.2
Backbone w/ our DSTS	12	88.4

9 Computational Cost of Inference

We compare the computational cost of conducting inference with DSTS on the Diving48 dataset using TPN as the backbone network. The results are shown in Table 5. Computational cost of inference for both **Backbone** and **Backbone w/ DSTS** refers to the time required to process a batch of samples. The experiments are conducted on Nvidia V100 GPU and the batch size is set to 8. Though using our proposed DSTS module leads to significant improvements in performance (see results in Table 1 and 2 of our main paper), the cost of inference only increases slightly.

Table 5. Computation cost of inference on Diving48 with TPN backbone. Note that, our method can bring obvious performance improvement (see Table 1 and 2 of our main paper).

Method	Computation cost (ms)
Backbone	741
Backbone w/ our DSTS	870

10 Details of Input Features of DSTS Module.

In our implementation, the feature maps before the last classification layer in TPN or Swin-B Transformer are used as the input of DSTS module. Specifically, for **TPN** backbone, the shape of input features for DSTS is $1024 \times 8 \times 16 \times 16$, and for the **Swin-B Transformer**, the shape of input features for DSTS is $1024 \times 16 \times 7 \times 7$.

References

1. Chen, Z., Li, Y., Bengio, S., Si, S.: You look twice: Gaternet for dynamic filter selection in cnns. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9172–9180 (2019) 1, 2
2. Kaiser, Ł., Bengio, S.: Discrete autoencoders for sequence models. arXiv preprint arXiv:1801.09797 (2018) 1, 2
3. Kaiser, L., Bengio, S., Roy, A., Vaswani, A., Parmar, N., Uszkoreit, J., Shazeer, N.: Fast decoding in sequence models using discrete latent variables. In: International Conference on Machine Learning. pp. 2390–2399. PMLR (2018) 1, 2