Supplement for “Searching for Two-Stream Models in Multivariate Space for Video Recognition”

1. Details of Search Algorithm

1.1. Background of PARSEC

In this section, we illustrate the adopted search algorithm PARSEC [2] in detail. Here we define a unique discrete architecture as A, which is sampled from a prior distribution $P(A|\boldsymbol{\alpha})$. Architecture parameters $\boldsymbol{\alpha}$ denote the probabilities of choosing different operations. The goal of PARSEC is to optimize the the architecture parameter $\boldsymbol{\alpha}$, in order to maximize the architecture accuracy. Concretely, for video recognition where we have video samples $\mathbf{X}$ and labels $\mathbf{y}$, probabilistic NAS can be formulated as optimizing the continuous architecture parameters α via an empirical Bayes Monte Carlo procedure [9]

$$\begin{aligned} P(\mathbf{y}|\mathbf{X},\omega,\alpha) &= \int P(\mathbf{y}|\mathbf{X},\omega,A)P(A|\boldsymbol{\alpha})\mathrm{d}A \\ &\approx \frac{1}{K}\sum_k P(\mathbf{y}|\mathbf{X},\omega,A_k), \end{aligned} \tag{1}$$

where ω denotes the model weights. The continuous integral of data likelihood is approximated by sampling K architectures and averaging the data likelihoods from them. We can jointly optimize architecture parameters $\boldsymbol{\alpha}$ and model weights ω by estimating gradients ∇_α log $P(\mathbf{y}|\mathbf{X},\omega,\boldsymbol{\alpha})$ and ∇_w log $P(\mathbf{y}|\mathbf{X},\omega,\boldsymbol{\alpha})$ through the sampled architectures. Typically, the number of sampled architecture K is set to 13, which is sufficient to search for a good architecture empirically, according to our preliminary experiments.

1.2. Cost-aware Search

Searching architecture without any efficiency constraint always leads to heavy architectures, which is not expected. Therefore, we further introduce an effective FLOPs constraint term $C(\boldsymbol{\alpha})$:

$$C(\boldsymbol{\alpha}) = \int \mathrm{cost}(A)P(A|\boldsymbol{\alpha})\mathrm{d}\alpha \approx \frac{1}{K}\sum_k \mathrm{cost}(A_k),$$

$$\mathrm{cost}(A_k) = \max(0, \frac{F(A_k)}{T_{\text{max}}-1}) + \max(0, 1-\frac{F(A_k)}{T_{\text{min}}}), \tag{2}$$

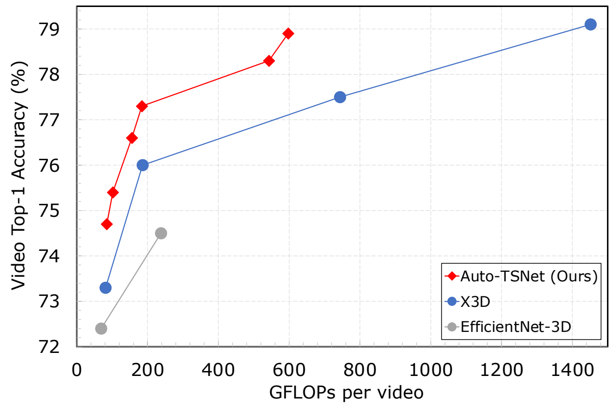

Figure 1: **Results of searched models on Kinetics-400.** We provide a zoom-in view of Figure 1 in our paper that focusing on the comparison of searched models.

where $[T_{min}, T_{max}]$ are the target architecture FLOPs range. $F(A_k)$ denotes the FLOPs of the sampled architecture A_k. The intuition here is to constraint the FLOPs of the searched architecture into the target range.

Therefore, the final objective function of our architecture search algorithm can be formulated as:

$$\min \mathcal{L}(\omega,\boldsymbol{\alpha}) = -\log P(\mathbf{y}|\mathbf{X},\omega,\boldsymbol{\alpha}) + \beta \log C(\boldsymbol{\alpha}), \tag{3}$$

where β is the coefficient of FLOPs constraint term.

In order to validate the effectiveness of the introduced FLOPs constraint term, we conducted experiments with different coefficient β, as shown in Table 1. The architecture is searched on Kinectics-100. We can observe that

Cost coefficient β	T_{min} (G)	T_{max} (G)	Architecture GFLOPs
0	-	-	2.76
0.05	1.37	1.43	1.93
0.1	1.37	1.43	1.56
0.2	1.37	1.43	1.39

Table 1: **Architecture search with different FLOPs constraint settings.**

the FLOPs of architecture is substantially larger than others when $\beta = 0$. When the cost coefficient β becomes large, the

Search Part	Epoch	Learning Rate	
		model weights	arch. param.
Sparse Stream	800	0.4	0.015
Dense Stream & Fusion Blocks	400	0.2	0.015
Attention Blocks	200	0.2	0.010

Table 2: **Architecture search settings.**

FLOPs of the searched architecture is gradually approaching to the pre-defined threshold range $[T_{min}, T_{max}]$.

2. Details of Searchable Two-Stream Fusion Block

In this section, we elaborate more the design of the fusion blocks we adopted in our search space. Inspired by [5], we adopted four candidate fusion operations. Here we define the feature map size of dense stream as $[C_D, T_D, H, W]$. $[C_S, T_S, H, W]$ for the sparse stream's feature map size. Each fusion operation will take the feature map from dense stream as input, and the output feature will be fused with sparse stream's feature map via summation.

Time-strided convolution. The temporal dimension of the input dense feature map is reduced from T_D to T_S by a 3D convolution, whose kernel size is 5×1^2. Then a point-wise convolution is applied to match the channel dimension to C_D.

Time-strided sampling. A uniform sampling is applied on temporal dimension, producing a feature map of $[C_D, T_S, H, W]$, followed by a point-wise convolution to change the channel dimension from C_D to C_S.

Time-to-channel. This fusion operation simply reshape the input feature to a feature of size $[\alpha C_D, T_S, H, W]$, where $\alpha = T_D/T_S$. An additional point-wise convolution is applied to match the channel dimension to C_S.

Skip. There's no fusion between the sparse and dense stream if "skip" operation is chosen.

3. Implementation Details

3.1. Searching Setting

We summarize the searching epoch and learning rate settings during architecture search in Table 2. Model weights are optimized using SGD optimizer and architecture parameters are optimized with Adam optimizer. We use momentum of 0.9, weight decay of $5e-5$ for the SGD optimizer and weight decay of 0 for the Adam optimizer. Dropout of 0.5 is used before the final classifier. Specifically, for the search of each part, we use the first 25% epochs to warm up the supernet, i.e., freezing the architecture parameters and only updating the model weights. We set mini-batch size to 8 per GPU with 32 GPUs, which means the total batch size is 256.

Model	Training Scale Jittering Range	Crop Size		
		Training	Evaluation	
			Center	*LCR*
Auto-TSNet-S	$[182, 228]$	160	160	182
Auto-TSNet-M	$[256, 320]$	224	224	256
Auto-TSNet-L	$[356, 446]$	312	312	356

Table 3: **The data processing for Auto-TSNet models.** Here *Center* and *LCR* denotes the *10-Center* and *10-LeftCenterRight* evaluation methods mentioned in our paper.

3.2. Training Setting

We adopt the open-source ClassyVision [1] for training, which is a PyTorch framework for video classification. Models are initialized with He initialization [7]. We use a SGD optimizer and set learning rate to 0.4 with a schedule of half-period cosine decaying [8]. A linear warm-up strategy [6] is also adopted for the first 34 epochs. We train a model for a total of 300 epochs (256 epochs for Auto-TSNet-L). The mini-batchsize is 8 clips per GPU with 32 GPUs, where the total batch size is set to 256. The dropout rate is set to 0.5 at the network head. We also apply AutoAugment [3] on each frame of input video clip. The data processing details are presented in Table 3, where each frame of the input video is obtained with a scale jittering followed by a random crop of a specified size.

3.3. Model Evaluation Setting

As illustrated in our paper, we use *10-Center* and *10-LeftCenterRight* evaluation methods to obtain our results. The evaluation crop sizes of Auto-TSNet models for both evaluation methods are shown in Table 3.

4. Model Complexity Benchmark

We also benchmark the the model complexity (Params, FLOPs & latency) and performance in Table. 4, for a better comparison with current SOTA model X3D [4]. The latency is bench-marked on a single NVIDIA A100 GPU with 500 repeat testing. We could observe that the proposed Auto-TSNet has comparable run-time latency compared with X3D, while surpassing it on the accuracy with a significant margin.

5. Random Search

We randomly sample 16 distinct architectures in our search space under the constraint that the 1-view GFLOPS is between $2.4G$ and $2.5G$. They are trained from scratch on Kinetics-400, and the results are shown in Figure. 2. Our searched Auto-TSNet-S model substantially outperforms randomly sampled architectures.

Model	Params (M)	Total Latency (s)	Total GFLOPs	Top-1 Acc (%)
X3D-S [4]	3.8	0.816	81	73.3
Auto-TSNet-S†(ours)	7.7	1.023	84	74.6
Auto-TSNet-S (ours)	8.6	1.206	102	**75.4**
X3D-M [4]	3.8	1.107	186	76.0
Auto-TSNet-M†(ours)	7.7	1.020	156	76.6
Auto-TSNet-M (ours)	8.6	1.215	183	**77.3**
X3D-L [4]	6.1	2.190	744	77.5
Auto-TSNet-L†(ours)	12.2	1.935	543	78.3
Auto-TSNet-L (ours)	13.2	2.133	597	**78.9**

Table 4: **Additional comparisons with other NAS models on Kinetics-400.** Auto-TSNet and X3D models are evaluated using 10-LeftCenterRight (30 views in total) testing. The latency and FLOPs reported in the table are calculated using 30 views. † denotes the model without attention blocks.

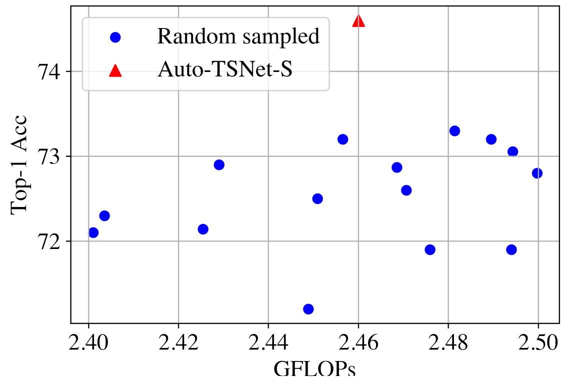

Figure 2: **Comparisons with random sampled architectures on Kinetics-400.**

References

[1] A. Adcock, V. Reis, M. Singh, Z. Yan, van der Maaten L., K. Zhang, S. Motwani, J. Guerin, N. Goyal, I. Misra, L. Gustafson, C. Changhan, and P. Goyal. Classy vision. 2019. 2

[2] Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019. 1

[3] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 2

[4] Christoph Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 203–213, 2020. 2, 3

[5] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 6202–6211, 2019. 2

[6] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 2

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 2

[8] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 2

[9] Carl Edward Rasmussen and Zoubin Ghahramani. Bayesian monte carlo. *Advances in neural information processing systems*, pages 505–512, 2003. 1