Temporal Action Detection Model Compression by Progressive Block Drop

Supplementary Material

In the supplementary material, we provide more details and more experimental results of our work. We organize the supplementary into the following sections.

- In Section A, we provide a detailed description of the hyperparameter settings used in our experiments.
- In Section B, we present detailed results on which layers are dropped at each iteration using the block drop method.
- In Section C, we employ the DETAD tool to conduct a comprehensive analysis of model performance across a broader range of metrics.
- In Section D, we investigate the data stability of our method through repeated experiments.
- In Section E, we explore the compatibility of the block drop method with sparse activations.
- In Section F, we further validate the effectiveness of our method through five ablation experiments.

A. Implementation Details

We use PyTorch 2.3.1 and 4 A800 GPUs for our experiments. The learning rate of LoRA is grid-searched within the range of 1e-5 to 1e-3, while the parameters of the other backbone components remain frozen. The hidden layer dimension of LoRA is fixed at one-quarter of the input dimension of the attention block. Other experimental details follow those described in [3].

B. Detailed Results of the Dropped Blocks

As discussed in Section 5.3, we applied our method to the THUMOS14 and ActivityNet-1.3 datasets. Here, we provide more detailed experimental results and configurations, including the specific blocks removed from each layer and the learning rate settings. As shown in Table A, the dropped blocks vary across models, demonstrating that the block drop selection evaluator can flexibly adapt to the characteristics of each model when selecting blocks to remove.

C. Quantitative Analysis with DETAD [1]

In order to assess the impact of our progressive block drop on detection performance, we conducted a detailed quantitative analysis of the pruned model using the DETAD [1]. Our analysis focuses on the false positive predictions before and after compression. As illustrated in Figure A, pruning notably improves localization, reducing the combined background and localization errors from 6.2+4.3 to 4.9+4.5. However, this improvement comes at the expense of classification accuracy, with the classification error increasing from 0.7 to 1.1.

Table A. The detailed dropped blocks for the experiments in Section 5.3. The dropped blocks vary across different models, indicating that our method is capable of adapting to the specific characteristics of each model.

Dataset	Backbone	Drop Blocks
THUMOS14	VideoMAE-S	[7]
		[7,10]
		[7,10,6]
		[7,10,6,3]
	VideoMAE-L	[20]
		[20,1]
		[20,1,4]
		[20,1,4,7]
		[20,1,4,7,6]
		[20,1,4,7,6,2]
ActivityNet-1.3	VideoMAE-S	[6]
		[6,3]
		[6,3,8]
		[6,3,8,11]

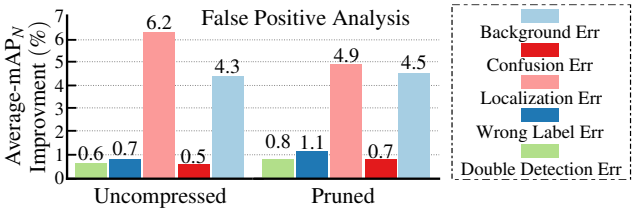

Figure A. Quantitative analysis between uncompressed and pruned model. The pruned model enhances localization performance while diminishing classification performance.

Furthermore, we used the DETAD tool to examine false negative predictions based on three metrics: **1) Coverage:** The proportion of an instance's duration relative to the video's total duration, divided into five categories. **2) Length:** The duration of an instance in seconds, categorized into five groups from short to extra-long. **3) Number of Instances:** The total count of same-class instances within a video, grouped into four categories. As shown in Figure B, the model with progressive block drop improved the detection of long-duration actions, reducing omission rates for actions with XL coverage from 8.9% to 6.7% and for XL length from 13.5% to 10.8%. This demonstrates the potential of our compression method in analyzing long-duration actions.

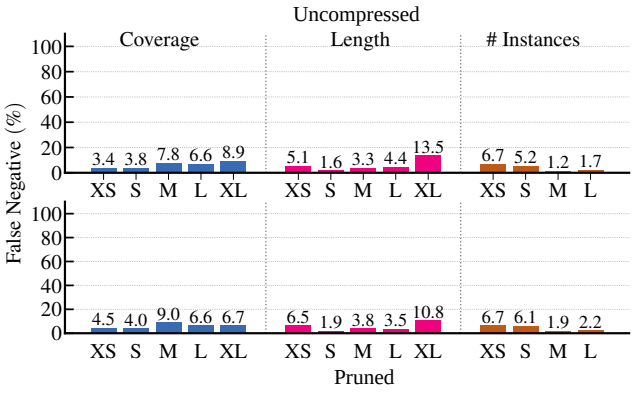

Figure B. Comparison of false negative analysis between uncompressed and pruned models.

D. Standard deviation of Results.

To compare the performance stability of our progressive block drop method with the random block drop approach. We conduct the experiment using three different random seeds to evaluate the performance of both methods: (1) the baseline approach where blocks are randomly dropped all at once, and (2) our progressive block drop method. The evaluation is performed on two datasets: THUMOS14 and ActivityNet-1.3. The experimental results are summarized in Table B. On THUMOS14, our method achieves a mean accuracy of 70.57% (±0.18), while randomly dropping blocks results in lower accuracy (69.06% and 68.41%). On ActivityNet-1.3, our method shows a slight improvement, with an accuracy of 37.74% (±0.03), compared to 37.57% for the random drop approach. This confirms the statistical significance and robustness of our method.

Table B. Comparison between ours and randomly dropping blocks. The results show that our method consistently achieves higher accuracy and more stable performance.

Drop Blocks ID	[2,5,9]	[0,4,7]	Ours
THUMOS14	68.41 (±0.06)	69.06 (±0.01)	70.57 (±0.18)
ActivityNet-1.3	37.57 (±0.08)	37.57 (±0.01)	37.74 (±0.03)

E. Combined with Sparse Activations.

Activation sparsity is another promising approach for achieving model acceleration [2]. In theory, it leverages the high density of zero values to accelerate subsequent computations by using sparse matrices. Given a model compressed using our method, we replace the GeLU activation function in each block with the ReLU function, following [4]. We then conduct experiments by fine-tuning this model using the alignment training method described in Section 4.2. As shown in Table C, after replacing the activation functions in all blocks with ReLU, the performance changed from 70.47% to 70.38%, which is essentially unchanged, while the activation sparsity increased from 0.18% to 83.42%. Therefore, it is entirely feasible to further accelerate the model's inference speed by combining specialized hardware and sparse matrix computation algorithms. This also demonstrates the strong scalability of our proposed progressive block drop method.

Table C. Results of our compatibility with sparse activation-based acceleration methods. Our pruned model can be further accelerated by sparse activation.

Model	Ours	Ours + Sparse Activation
mAP	70.47	70.38
Sparsity	0.18%	**83.42%**

Table D. Comparison of three metrics in the block drop selection evaluator. Training loss and mAP benefits more on block selection.

Dataset	Uncompressed	Drop Metric		
		Train Loss	mAP	MSE
THUMOS14	70.43	**70.88**	70.47	69.37
ActivityNet-1.3	37.75	37.71	**37.77**	37.25

F. Ablation Studies

F.1. Different Choices of Block Drop Criteria

We compare different importance metrics to select which blocks to drop. We employed three metrics: **1) Train Loss**: the average loss over all data when training the subnet after dropping blocks. **2) MSE**: perform inference on the training set and compute the feature differences before and after each block. The average MSE over all input data is used as the evaluation metric. **3) mAP**: performance of the subnet evaluated on the training set. From Table D, when using mAP to select blocks to drop, the performance on both datasets surpasses that of the uncompressed model (at least $\uparrow$ 0.02%). While the train loss metric does not exceed the uncompressed model on the ActivityNet-1.3 dataset, it is relatively close ($\downarrow$ 0.04%) and can outperform the uncompressed model on the THUMOS14 dataset. However, MSE performs worse on both datasets (at least $\downarrow$ 0.50%). Therefore, we choose mAP as the evaluation metric due to its more stable performance across different datasets.

F.2. Ablation on the Effect of Fine-Tuning After Block Dropping

Our experiments underscore the critical importance of fine-tuning following block dropping. In our ablation studies, as illustrated in Table E, pruning three blocks without any fine-tuning led to a marked decline in mAP ($\downarrow$ 5.14%). In contrast, when fine-tuning was applied, the mAP was effectively restored to 70.47%. This recovery highlights that fine-tuning is essential to bridge the domain gap between the pre-trained model and the target TAD dataset, thereby

Table E. Effect of fine-tuning (FT) after block dropping. FT after block pruning is essential for restoring performance.

Drop #Block	Baseline	1	2	3
without FT	70.43	69.21 (↓ **1.22**)	67.13 (↓ **3.30**)	65.29 (↓ **5.14**)
with FT	70.43	71.06 (↑ **0.63**)	71.37 (↑ **0.94**)	70.47 (↑ **0.04**)

Table F. Comparison between LoRA and full fine-tuning. Param refers to the trainable parameters within the backbone. The use of LoRA technology can effectively reduce computational costs while benefiting model performance.

Drop #Block	Full Fine-tuning		LoRA	
	mAP	Param (M)	mAP	Param (M)
Baseline	70.43	20.86	70.43	3.11
1	70.67 (↑ **0.24**)	19.17	71.06 (↑ **0.63**)	2.90
2	70.75 (↑ **0.32**)	17.48	71.37 (↑ **0.94**)	2.68
3	70.42 (↓ **0.01**)	15.79	70.47 (↑ **0.04**)	2.47

mitigating the adverse effects of structural modifications.

F.3. Ablation on the Impact of LoRA

To further enhance the adaptation process while reducing computational overhead, we integrate Low-Rank Adaptation (LoRA) into our fine-tuning strategy. As presented in Table F, the incorporation of LoRA achieves comparable—and in some cases superior—accuracy relative to full fine-tuning, yet requires significantly fewer trainable parameters. For example, when pruning two blocks, the LoRA-augmented approach attained an mAP improvement of 0.94% with only 2.68M tunable parameters, in contrast to 17.48M when employing full fine-tuning. These results validate that LoRA not only minimizes the computational cost of fine-tuning but also contributes positively to performance, confirming its practical utility in our framework.

F.4. Ablation on Selective Fine-Tuning

We further explored the efficiency of our adaptation strategy by freezing the detection head and all network components except the LoRA modules within the backbone. This approach resulted in a modest mAP decrease of 1.57% (from 70.47% to 68.90%) when three blocks were pruned. Although selective fine-tuning of only the LoRA blocks considerably reduces the number of trainable parameters, the performance drop suggests that jointly fine-tuning both the backbone and the detection head is preferable for optimal performance recovery. These findings indicate that while freezing the detection head can offer additional computational savings, a comprehensive fine-tuning strategy remains essential to fully recover the performance.

F.5. Ablation on Loss Functions for Alignment

To investigate the effect of different alignment losses on the performance recovery, we design several ablation experiments: **1) no alignment loss**; **2) action recognition and localization alignment**; **3) cross-depth feature alignment**. The experimental results in Table G, demonstrate that the best performance is achieved when both alignment techniques are used simultaneously. This indicates that combining these two losses more effectively guides the model to learn in a way that is closer to the uncompressed model.

Table G. Ablation on different alignment losses on THUMOS14. The combination of both alignment losses yields the best result.

Alignment Level					
	Feature		✓		✓
	Prediction			✓	✓
mAP (tIoU=0.5)		69.25	70.23	70.39	**70.47**

References

[1] Humam Alwassel, Fabian Caba Heilbron, Victor Escorcia, and Bernard Ghanem. Diagnosing error in temporal action detectors. In *Proceedings of the European conference on computer vision (ECCV)*, pages 256–272, 2018. 1

[2] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning (ICML)*, pages 5533–5543. PMLR, 2020. 2

[3] Shuming Liu, Chen-Lin Zhang, Chen Zhao, and Bernard Ghanem. End-to-end temporal action detection with 1b parameters across 1000 frames. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18591–18601, 2024. 1

[4] Seyed Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. 2