Automated Progressive Learning for Efficient Training of Vision Transformers
Supplementary Material

Changlin Li[1,2,3] Bohan Zhuang[3†] Guangrun Wang[4] Xiaodan Liang[5] Xiaojun Chang[2] Yi Yang[6]
[1]Baidu Research [2]ReLER, AAII, University of Technology Sydney
[3]Monash University [4]University of Oxford [5]Sun Yat-sen University [6]Zhejiang University
changlinli.ai@gmail.com, bohan.zhuang@monash.edu, wanggrun@gmail.com,
xdliang328@gmail.com, xiaojun.chang@uts.edu.au, yangyics@zju.edu.cn

A. Definition of Compared Growth Operators

Given a smaller network ψ_s and a larger network ψ_ℓ, a growth operator ζ maps the parameters of the smaller one ω_s to the parameters of the larger one ω_ℓ by: $\omega_\ell = \zeta(\omega_s)$. Let ω_ℓ^i denotes the parameters of the i-th layer in ψ_ℓ[1]. We consider several ζ in depth dimension that maps ω_s to layer i of ψ_ℓ by: $\omega_\ell^i = \zeta(\omega_s, i)$.

RandInit. *RandInit* copies the original layers in ψ_s and random initialize the newly added layers:

$$\zeta_{\mathit{RandInit}}(\omega_s, i) = \begin{cases} \omega_s^i, & i \le l_s \\ \mathit{RandInit}, & i > l_s. \end{cases} \tag{1}$$

Stacking. *Stacking* duplicates the original layers and directly stacks the duplicated ones on top of them:

$$\zeta_{\mathit{Stacking}}(\omega_s, i) = \omega_s^{i \bmod l_s}. \tag{2}$$

Interpolation. *Interpolation* interpolates new layers of ψ_ℓ in between original ones and copy the weights from their nearest neighbor in ψ_s:

$$\zeta_{\mathit{Interpolation}}(\omega_s, i) = \omega_s^{\lfloor i/l_s \rfloor}. \tag{3}$$

B. Implementation Details

Our ImageNet training settings follow closely to the original training settings of DeiT [10] and VOLO [13], respectively. We use the AdamW optimizer [9] with an initial learning rate of 1e-3, a total batch size of 1024 and a weight decay rate of 5e-2 for both architectures. The learning rate decays following a cosine schedule with 20 epochs warm-up for VOLO models and 5 epochs warm-up for DeiT models. For both architectures, we use exponential moving average with best momentum factor in $\{0.998, 0.9986, 0.999, 0.9996\}$.

For DeiT training, we use RandAugment [2] with 9 magnitude and 0.5 magnitude std., mixup [15] with 0.8 probability, cutmix [14] with 1.0 probability, random erasing [16] with 0.25 probability, stochastic depth [7] with 0.1 probability and repeated augmentation [6].

For VOLO training, we use RandAugment [2], random erasing [16], stochastic depth [7], token labeling with MixToken [8], with magnitude of RandAugment, probability of random erasing and stochastic depth adjusted by Adaptive Regularization.

Adaptive Regularization. The detailed settings of Adaptive Regularization for VOLO progressive training is shown in Tab. I. These hyper-parameters are set heuristically regarding the model size. They perform fairly well in our experiments, but could still be sub-optimal.

Regularization	D0		D1	
	min	max	min	max
RandAugment [2]	4.5	9	4.5	9
Random Erasing [16]	0	0.25	0.0625	0.25
Stoch. Depth [7]	0	0.1	0.1	0.2

Table I. Adaptive Regularization Settings (magnitude of RandAugment [2], probability of Random Erasing [16] and Stochastic Depth [7]) for progressive training of VOLO models.

Growth Space Λ_k in Each Stage. We find emprically that the elastic supernet converges faster when the number of sub-networks are smaller. Thus, restricting the growth space Λ_k in each stage could help the convergence of the supernet. In practice, we make the restriction that $|\Lambda_k| \le 9$. Specifically, in the first stage, we use the largest, the smallest and the medium candidates of n and l in Ω to construct Λ_1, which makes it possible to route to the whole network and perform regular training if the growing "ticket" (suitable sub-network) does not exist. In each of the following stages, we include the next 3 candidates of l and the next 1 candidate of n, forming a growth space with $2 \times 4 = 8$ candidates.

†Corresponding author.
[1]In our default setting, i begins from the layer near the classifier.

Model	Training scheme	FLOPs (avg. per step)	Speedup	Runtime (GPU Hours)	Speedup	Top-1 (%)	Top-1@288 (%)
100 epochs							
DeiT-S [10]	Original	4.6G	-	71	-	74.1	74.6
	Prog	2.4G	+91.6%	46	+53.6%	72.6	73.2
	AutoProg	2.8G	+62.0%	50	+40.7%	**74.4**	**74.9**
VOLO-D1 [13]	Original	6.8G	-	150	-	82.6	83.0
	Prog	3.7G	+84.7%	93	+60.9%	81.7	82.1
	AutoProg 0.5Ω	3.3G	+104.2%	91	+65.6%	**82.8**	**83.2**
	AutoProg 0.4Ω	2.9G	**+132.2%**	81	**+85.1%**	82.7	83.1
VOLO-D2 [13]	Original	14.1G	-	277	-	83.6	84.1
	Prog	7.5G	+87.7%	180	+54.4%	82.9	83.3
	AutoProg	8.3G	+68.7%	191	+45.3%	**83.8**	**84.2**
300 epochs							
DeiT-Tiny [10]	Original	1.2G	-	144	-	72.2	72.9
	AutoProg	0.7G	+82.1%	95	+51.2%	**72.4**	**73.0**
DeiT-S [10]	Original	4.6G	-	213	-	79.8	80.1
	AutoProg	2.8G	+62.0%	150	+42.0%	79.8	80.1
VOLO-D1 [13]	Original	6.8G	-	487	-	84.2	84.4
	AutoProg	4.0G	+68.9%	327	+48.9%	**84.3**	**84.6**
VOLO-D2 [13]	Original	14.1G	-	863	-	85.2	85.1
	AutoProg	8.8G	+60.7%	605	+42.7%	85.2	**85.2**

Table II. Detailed results of efficient training on ImageNet. Best results are marked with **Bold**; our method or default settings are highlighted in purple . Top-1@288 denotes Top-1 Accuracy when directly testing on 288×288 input size, *without* finetuning. Runtime is rounded to integer.

C. Additional Results

Theoretical Speedup. In Tab. II, we calculate the average FLOPs per step of different learning schemes. AutoProg consistently achieves more than 60% speedup on theoretical computation. Remarkably, VOLO-D1 trained for 100 epochs with AutoProg 0.4Ω achieves **132.2%** theoretical acceleration. The gap between theoretical and practical speedup indicates large potential of AutoProg. We leave the further improvement of practical speedup to future works; for example, AutoProg can be further accelerated by adjusting the batch size to fill up the GPU memory during progressive learning.

Comparison with Progressively Stacking. Progressively Stacking [3] (ProgStack) is a popular progressive learning method in NLP to accelerate BERT pretraining. It begins from $\frac{1}{4}$ of original layers, then copies and stacks the layers twice during training. Originally, it has three training stages with number of steps following a ratio 5:7:28. In CompoundGrow [4], this baseline is implemented as three stages with 3:4:3 step ratio. Our implementation follows closer to the original paper, using a ratio of 1:2:5. The results are shown in Tab. III. ProgStack achieves relatively small speedup with performance drop (0.4%). Our MoGrow reduces this performance gap to 0.1%. AutoProg achieves 74.1% more speedup and 0.5% accuracy improvement over the ProgStack baseline.

Training scheme	Runtime (GPU hours)	Speedup	Top-1 (%)
Baseline	150.2	-	82.6
ProgStack [3]	135.3	+11.0%	82.2
+ MoGrow	136.0	+10.4%	82.5
Prog	93.3	+60.9%	81.7
AutoProg 0.4Ω	81.1	**+85.1%**	**82.7**

Table III. Comparison with progressively stacking.

Combine with AMP. Automatic mixed precision (AMP) [52] is a successful and mature low-bit precision efficient training method. We conduct experiments to prove that the speed-up achieved by AutoProg is orthogonal to that of AMP. As shown in Tab. IV, the relative speed-up achieved by AutoProg with or without AMP is comparable (+85.1% *vs*. +87.5%), proving the orthogonal speed-up.

Method	Speed-up	Top-1 Acc. (%)
Original (w/o AMP)	-	82.6
AMP	+74.0%	82.6
AutoProg	**+87.5%**	**82.7**
AMP + AutoProg	**+222.1%** (**+85.1%** over AMP)	**82.7**

Table IV. Speed-up of AutoProg is orthogonal to AMP [52].

Number of stages. We perform experiments to analyze the impact of the number of stages on AutoProg with different initial scaling ratios (0.5 and 0.4). As shown in Tab. V, AutoProg is not very sensitive to stage number settings. Fewer than 4 yields more speed-up, but could damage the performance. In general, the default 4 stages setting performs the best. When scaling the stage number to 50, there are only supernet training phases (2 epochs per stage) during the whole 100 epochs training, causing severe performance degradation.

Ratio	Num. Stages	Orig.	3	4	5	50
0.5	Speed-up	-	**+69.1%**	+65.6%	+63.6%	+48.5%
	Top-1 Acc. (%)	82.6	82.6	**82.8**	**82.8**	81.7
0.4	Speed-up	-	+90.8%	**+85.1%**	+80.4%	-
	Top-1 Acc. (%)	82.6	82.4	**82.7**	**82.7**	-

Table V. Ablation analysis on number of stages.

Effect of Progressive Learning in AutoProg. AutoProg is comprised by its two main components, "Auto" and "Prog". The effectiveness of "Auto" is already studied by comparing with Prog in the main text. Here, we study the effectiveness of progressive learning in AutoProg by training an elastic supernet baseline for 100 epochs without progressive growing to compare with AutoProg. Specifically, we treat VOLO-D1 as an Elastic Supernet, and train it by randomly sampling one of its sub-networks in each step, same to the search stage in AutoProg. The results are shown in Tab. VI. In previous works that uses elastic supernet [1, 11, 12], the supernet usually requires more training iterations to reach a comparable performance to a single model. As expected, the supernet performance is lower than the original network given the same training epochs. Specifically, AutoProg improves over elastic supernet baseline by 1.1% Top-1 accuracy, with 17.1% higher training speedup, reaching the performance of the original model with the same training epochs but much faster, which proves the superiority of progressive learning.

Method	Speedup	Top-1 Acc. (%)
Original	-	82.6
Supernet	48.5%	81.7
AutoProg	**65.6%**	**82.8**

Table VI. Ablation analysis of progressive learning in AutoProg with VOLO-D1.

Analyse of Searched Growth Schedule. Two typical growth schedules searched by AutoProg are shown in Tab. VII. AutoProg clearly prefers smaller token number than smaller layer number. Nevertheless, selecting a small layer number in the first stage is still a good choice, as both of the two schemes use reduced layers in the first stage.

Stage k		1	2	3	4
VOLO-D1 100e 0.4Ω	l	0.4	1	1	1
	n	0.4	0.6	0.6	1
VOLO-D2 300e	l	0.83	1	1	1
	n	0.5	0.67	0.83	1

Table VII. Searched growth schedules for VOLO-D1 0.4Ω, 100 epochs, and VOLO-D2, 300 epochs.

Retraining with Searched Growth Schedule. To evaluate the searched growth schedule, we perform retraining from scratch with VOLO-D1, using the schedule searched by AutoProg 0.4Ω. As shown in Tab. VIII, retraining takes slightly longer time (-0.6% speedup) because the speed of searched optimal sub-networks could be slightly slower than the average speed of sub-networks in the elastic supernet. Retraining reaches the same final accuracy, proving that the searched growth schedule can be used separately.

Training scheme	Runtime (GPU hours)	Speedup	Top-1 (%)
Baseline	150.2	-	82.6
AutoProg 0.4Ω	81.1	**+85.1%**	**82.7**
Retrain	81.4	+84.5%	**82.7**

Table VIII. Retraining results with searched growth schedule on VOLO-D1, 100 epochs.

Extend to CNNs. To explore the effect of our policy on CNNs, we conduct experiments with ResNet50 [5], and found that the policy searched on ViTs generalizes very well on CNNs (see Tab. IX). These results imply that AutoProg opens an interesting direction (automated progressive learning) to develop more general learning methods for a wide computer vision field.

Method	Speed-up	Top-1 Acc. (%)
Original	-	77.3
AutoProg	**+56.9%**	77.3

Table IX. AutoProg with ResNet50 [5] on ImageNet (100 epochs).

References

[1] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *ICCV*, 2021. 3

[2] Ekin Dogus Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPRW*, 2020. 1

[3] Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tie-Yan Liu. Efficient training of bert by progressively stacking. In *ICML*, 2019. 2

[4] Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. On the transformer growth for progressive bert training. In *NAACL*, 2021. 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3

[6] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *CVPR*, 2020. 1

[7] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 1

[8] Zihang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Yujun Shi, Xiaojie Jin, Anran Wang, and Jiashi Feng. All tokens matter: Token labeling for training better vision transformers. *arXiv preprint arXiv:2104.10858*, 2021. 1

[9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 1

[10] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 1, 2

[11] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, T. Huang, Xiaodan Song, and Quoc V. Le. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, 2020. 3

[12] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *ICLR*, 2019. 3

[13] Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. VOLO: Vision outlooker for visual recognition. *arXiv preprint arXiv:2106.13112*, 2021. 1, 2

[14] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Young Joon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019. 1

[15] Hongyi Zhang, Moustapha Cissé, Yann Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 1

[16] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020. 1