Supplementary Material for Stitchable Neural Networks

Zizheng Pan Jianfei Cai Bohan Zhuang[†]

ZIP Lab, Monash University

https://snnet.github.io

We organize our supplementary material as follows.

- In Section A, we provide further explanation of the proposed nearest stitching strategy.
- In Section B, we study the effect of different sizes and strides of sliding windows for stitching.
- In Section C, we study the effect of different training epochs.
- In Section D, we show the effectiveness of our training strategy by comparing with sandwich sampling rule and inplace distillation [7].
- In Section E, we discuss the effect of training without the pretrained weights of anchors.
- In Section F, we experiment with different number of samples for initializing stitching layers.
- In Section G, we provide additional discussion with One-shot NAS.
- In Section H, we compare SN-Net with LayerDrop [1] at inference time.

A. Detailed Illustration of Nearest Stitching Strategy

In the proposed SN-Net, we introduce a nearest stitching strategy which limits the stitching between two anchors of the nearest complexity/performance. In Figure A, we describe more details for this approach based on DeiT [6]. Under the nearest stitching, we limit the stitches to two types: Ti-S and S-B, which connects DeiT-Ti/S and DeiT-S/B, respectively. Experiments in the main manuscript have shown that stitching anchors with a larger complexity/performance gap or sequentially stitching more than two anchors achieves inferior performance.

[†]Corresponding author. E-mail: bohan.zhuang@gmail.com

B. Effect of Different Sizes and Strides of Sliding Windows

We explore different settings of sliding windows in SN-Net. In Figure B, we visualize the results of using different kernel sizes and strides in stitching DeiT models. Overall, different settings can produce a different number of stitches but achieve similar good performance. However, it is worth noting that within a larger window, a stitching layer needs to map activations with more dissimilar representations, which potentially results in some bad-performed stitches, as shown in the case of $k = 4, s = 4$ in Figure B.

C. Effect of Different Training Epochs

In our default setting, we train DeiT-based SN-Net with 50 epochs. In Figure C, we show that even with only 15 epochs, many stitches in SN-Net still perform favorably. With more training epochs, we observe consistent performance gain for all stitches, especially for those close to the smaller anchors, *e.g.* stitches around DeiT-Ti and DeiT-S. This is also reasonable as these stitches perform badly at the very beginning, thus particularly requiring more training time to obtain good performance.

D. Effect of Training Strategy

To train SN-Net, we adopt a simple training strategy by randomly sampling a stitch at each training iteration and using simple distillation for all stitches with a typical teacher (*e.g.*, RegNetY-160 [5]). To show the effectiveness of this strategy, we conduct experiments by training SN-Net with sandwich sampling rule and inplace distillation [7], which is a common practice for training supernets in NAS [3,4,8]. Specifically, we simultaneously sample one stitch and its connected pair of anchors at each training iteration. At the same time, we use the larger anchor as the teacher to guide the smaller anchor and the sampled stitch. However, as shown in Figure D, this approach mainly improves the smaller anchors (*i.e.*, DeiT-Ti/S) while most stitches cannot outperform those under our training strategy. It is also worth noting that the sandwich rule requires intensive memory/time cost due to training multiple networks at one train-

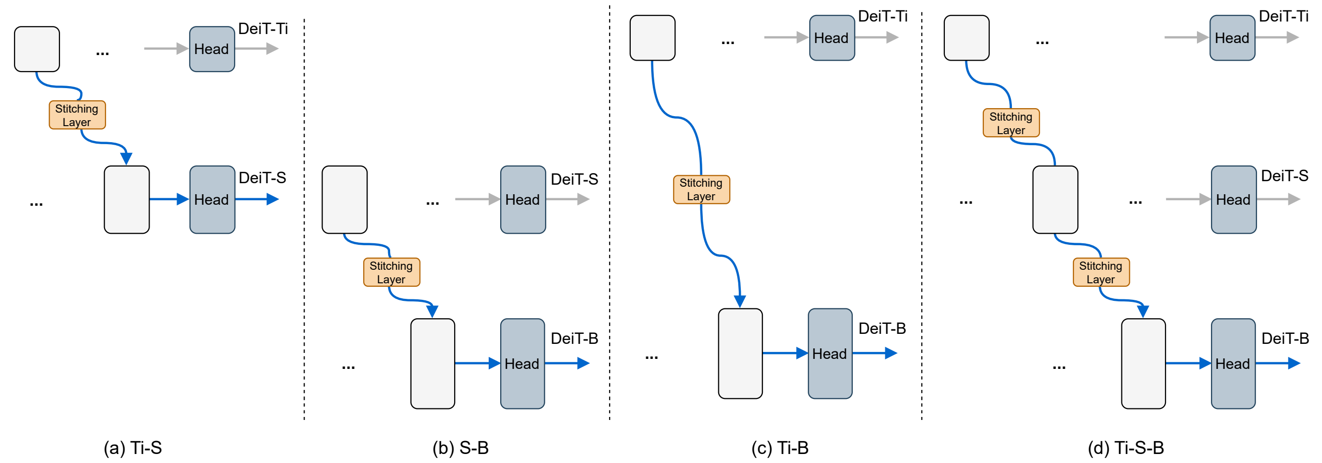

Figure A. Four types of stitches based on DeiT-Ti/S/B. Under the proposed nearest stitching strategy, we limit the stitching between two anchors of the nearest model complexity/performance, *i.e.*, Figure (a) and (b), while excluding stitching anchors with a larger complexity/performance gap (Figure (c)) or sequentially stitching more than two anchors (Figure (d)).

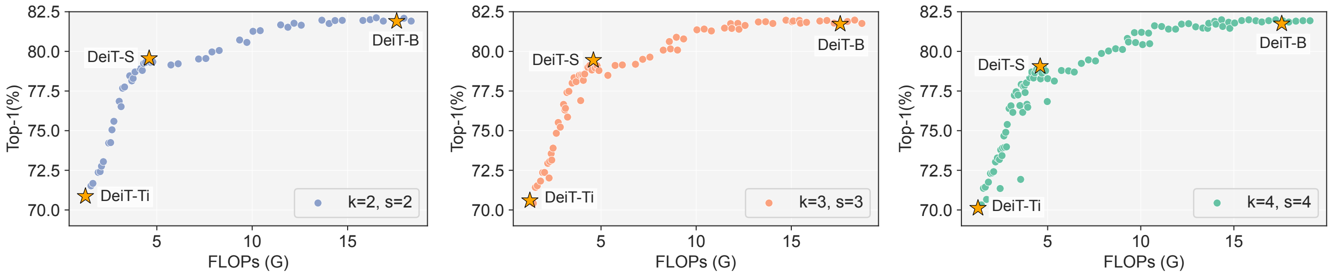

Figure B. Effect of different sizes of sliding windows. k and s refer to the kernel size and stride for controlling the sliding windows. From left to right, the kernel sizes and strides of 2, 3 and 4 produce 51, 75 and 99 stitches, respectively.

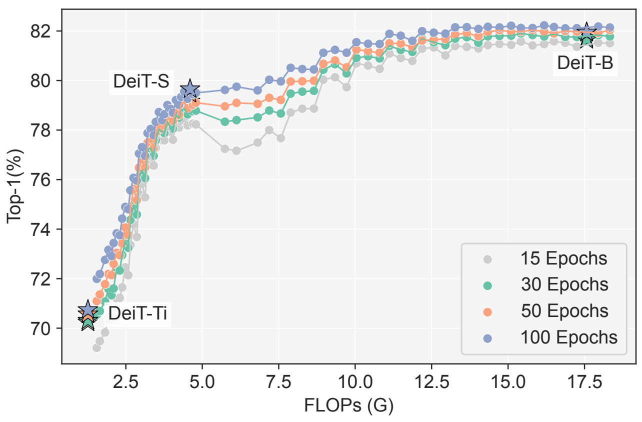

Figure C. Effect of different training epochs.

ing iteration. In contrast, ours requires a similar training cost for each training iteration as a normal network training [6].

E. Training without Pretrained Weights

The foundation of SN-Net is based on the pretrained model families in the large-scale model zoo. Without the pretrained weights of anchors, we find SN-Net failed to converge (training failed within 10 epochs based on the default experiment settings of DeiT), which aligns with our assumption that pretrained anchors help to reduce many training difficulties, such as the interference among stitches.

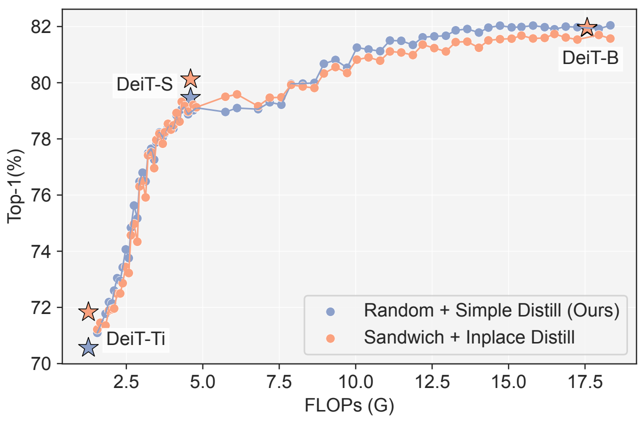

Figure D. Comparison between our training strategy and common supernet training strategy in NAS (*i.e.*, sandwich sampling rule and inplace distillation [7]).

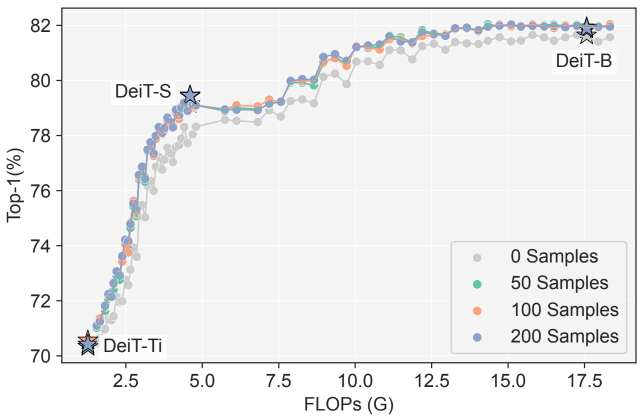

Figure E. Effect of different number of samples for initializing stitching layers. With 0 samples, the initialization is equivalent to the default Kaiming initialization in PyTorch.

F. Effect of Different Number of Samples for Initializing Stitching Layers

By default, we randomly sample 100 training images to initialize the stitching layers in SN-Net. To explore the effect of different number of samples for initialization, we train DeiT-based SN-Net by using 50, 100 and 200 training images on ImageNet with the same 50 epochs of training. As shown in Figure E, although all settings achieve better performance than the default Kaiming initialization in PyTorch, we find that using more samples does not bring more performance gain. Besides, since solving the least-squares solution with more samples can increase the memory cost at the beginning of training, we set the default number of samples for initializing stitching layers to 100 to avoid the potential "out of memory" issue.

G. Compared with One-shot NAS

As discussed earlier, SN-Net is fundamentally different from one-shot NAS. Specifically, one-shot NAS trains a supernet from scratch and searches for an optimal subnetwork during deployment to meet a specific resource constraint with complicated techniques (*e.g.*, evolutionary search) and expensive cost (*e.g.*, $> 2K$ GPU hours in [8]). In contrast, SN-Net aims to cheaply and fast assemble pretrained model families (*e.g.*, ∼110 GPU hours) to get a scalable network, and instantly select optimal stitches due to the interpolation effect. In our experiments, we use DeiTs and Swins as two examples to show that SN-Net is a universal framework. Besides, we show in Figure F that we easily achieve comparable performance with BigNASModel-XL [8] (80.7% *vs*. 80.9%) with lower FLOPs (977M *vs*. 1040M) by stitching LeViTs [2].

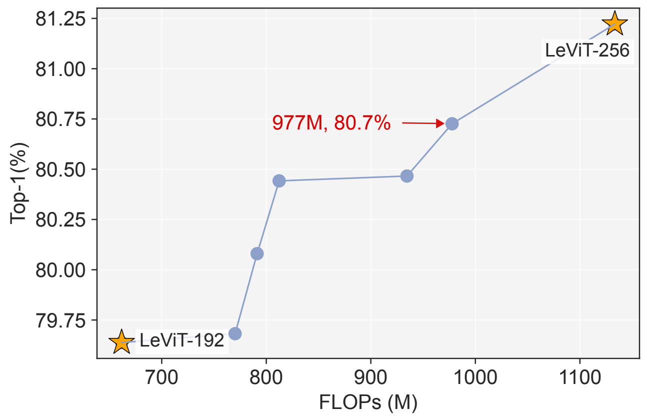

Figure F. Stitching LeViT-192 and LeViT-256

H. Compared with LayerDrop at Inference Time

LayerDrop [1] is a form of structured dropout which randomly drops Transformer layers during training for regularization. It also facilitates efficient pruning by dropping some layers at inference time. In DeiT-based SN-Net, the anchors are already pretrained with a drop rate of 0.1. To show the advantage of our method, we train DeiT-B (*i.e.*, the largest model in DeiT family) with a more aggressive path drop rate (0.5) and achieve 81.4% Top-1 accuracy on ImageNet. However, cropping some layers of this trained network during testing performs badly, *e.g.*, throwing the first 6 blocks (achieving 0.2%), the last 6 blocks (52.7%), and every other (72.7% with 8.9G FLOPs), while our method achieves 72.6% with 2.1G FLOPs.

References

[1] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *ICLR*, 2020. 1, 3

[2] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *ICCV*, pages 12259–12269, 2021. 3

[3] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *CVPR*, pages 8607–8617, June 2021. 1

[4] Huaijin Pi, Huiyu Wang, Yingwei Li, Zizhang Li, and Alan L. Yuille. Searching for trionet: Combining convolution with local and global self-attention. In *BMVC*, page 141, 2021. 1

[5] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, pages 10425–10433, 2020. 1

[6] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, pages 10347–10357, 2021. 1, 2

[7] Jiahui Yu and Thomas S. Huang. Universally slimmable networks and improved training techniques. In *ICCV*, pages 1803–1811, 2019. 1, 2

[8] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas S. Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, pages 702–717, 2020. 1, 3