A. Training Hyperparameters

A.1. Hyperparameters for Single-trial Model Scaling.

Our training hyperparameters are the same as DeiT-B as given in Table 1. We find using repeat augmentation and erasing augmentation doesn't show any performance improvement. As such, we do not use them in the training phase.

A.2. Hyperparameters for NAS

The regularized evolution algorithm discussed in Sec.4 is identical to AmoebaNet. We set the population size set to 50 and the tournament size set to 10. For the reward function, exponent $\epsilon = -0.07$, the FLOPs target is set to $FLOPs_0 = 10000M$.

B. Details of Baseline Scaling Operators

B.1. Learning Curve of Scaling Operators

The learning curve for different expansion operators is given in Figure 2. The γ_{ST} with momentum information outperforms other baselines.

B.2. Details Explanations for bert2BERT and Learn-to-share

bert2BERT (γ_{b2B}): We use a simple example to illustrate the key idea of bert2BERT here. Assuming we are expanding the first layer w_0 and the input feature vector is d_{in}, the corresponding output of the dense layer would be $d_o = d_{in}^T w_0$. w_0 has a dimension of 2×2 and d_{in} has a dimension of 2×1. After layer scaling, w_0'' has a size of 4×4.

$$d_{in} = \begin{bmatrix} a \\ b \end{bmatrix}, \quad w_0 = \begin{bmatrix} o & p \\ q & r \end{bmatrix}, \quad d_o^T = d_{in}^T w_0 = \begin{bmatrix} a_o \\ b_o \end{bmatrix} \tag{1}$$

When expanding the weight matrix w_0 given in Eq.1 from a 2×2 matrix into a 4×4 matrix, we first expand the input dimensions.

We randomly select two rows, e.g., the first row, and duplicate them. Then, we normalize these rows based on the number of duplications. The corresponding input features will be duplicated in the same fashion without normalization. The result dense layers are given as follows:

$$d'_{in} = \begin{bmatrix} a \\ b \\ a \\ a \end{bmatrix}, \quad w'_0 = \begin{bmatrix} \frac{o}{3} & \frac{p}{3} \\ q & r \\ \frac{o}{3} & \frac{p}{3} \\ \frac{o}{3} & \frac{p}{3} \end{bmatrix} \tag{2}$$

As is shown above, the expanded output ${d'_{in}}^T w'_0 = d_{in}^T w_0$ does not change during the expansion.

Next, we randomly select two columns, e.g., the second column, and duplicate it without normalization.

$$d''_{in} = \begin{bmatrix} a \\ b \\ a \\ a \end{bmatrix}, \quad w''_0 = \begin{bmatrix} \frac{o}{3} & \frac{p}{3} & \frac{p}{3} & \frac{p}{3} \\ q & r & r & r \\ \frac{o}{3} & \frac{p}{3} & \frac{p}{3} & \frac{p}{3} \\ \frac{o}{3} & \frac{p}{3} & \frac{p}{3} & \frac{p}{3} \end{bmatrix} \tag{3}$$

The final output (${o''} = {d''_{in}}^T w''_0$) would be $o'' = [a_o, b_o, b_o, b_o]$. For the following layer w_1, the input is determined and thus the policy of row duplication is determined as well. For w_1, we continue the same procedure for expanding columns (i.e., randomly select columns and duplicate them). And so on, the model functionality can be preserved.

$$LayerNorm(o) = \frac{(o'' - \mu_o)}{\sigma_o} \odot W^{LN} + b^{LN} \tag{4}$$

However, if the next layer is LayerNorm (Eq.4). The mean (μ_o) and variance (σ_o) of the output o changes. $\odot$ denotes the element-wise multiplication. During expansion, we don't know the relationship between a_o and b_o, so bert2BERT cannot preserve functionality through changing the LN scale and LN-bias, i.e. W^{LN} and b^{LN}.

On the other hand, γ_{ST} will yield output $o'' = [a_o, b_o, a_o, b_o]$. The mean μ_o and the variance σ_o of the output vector does not change.

Learn-to-grow. learn-to-grow proposes to learn linear matrices that map the pretrained weights into larger weight matrices to preserve the functionality of the small pretrained model. We denote its width and depth expansion operator as γ_{ltg} and β_{ltg}, respectively.

$$W'_i = \gamma_{ltg}(w_i) = H_i w_i H_i^T, \quad i \in \{1, ..., l\} \tag{5}$$

Here, H_i ($D \times d$) is a trainable linear layer that maps the dense layer w_i into W'_i. w_i has a dimension of $d \times d$ and W'_i has a size of $D \times D$. For layer normalization and weight bias with a dimension of $d \times 1$, the expansion is similar to Eq 5.

After width expansion, learn-to-share trains another set of linear mappings for depth expansion that expands W' into W:

$$W_i = \beta_{ltg}(w_i) = \Sigma_{j=1}^{l} P_{i,j} W'_j, \quad i \in \{1, ..., L\} \tag{6}$$

Here P_i is a $1 \times l$ vector. l is the number of layers in the pretrained model; L is the number of layers in the scaled model. This means the expanded layer W_i is the weighted sum of W'_j where $j \in \{1, ..., l\}$.

Table 1. Hyperparameters for model scaling experiments. The hyperparameters are identical to DeiT-B. We find batch augmentation and Erasing are not useful to increase the final task accuracy.

Search method	Search method	Learning rate decay	Warmup epoch	Label smoothing	Dropout	Drop path	Repeat Aug	Gradient clip	RandAug	Mixup	Cutmix	Erasing
4096	4e-3	cosine	5	0.1	0.0	0.1	×	×	✓	✓	✓	✓

The linear mappings (H, P) are introduced to scale every dense layer in the scaled ViT. These mappings contain a large number of parameters and require a prohibitively expensive hardware memory for training. Some techniques are proposed in the paper to reduce the number of parameters, such as Kronecker factorization.

In this paper, we find the objective of training these linear mappings is the same as training the scaled model (Eq.5). For ViT-S→B, learn-to-grow can achieve 72% initial accuracy. Specifically, learn-to-grow trains the linear mapping H, P for around 200 steps and scale the model according to Eq.5-6. However, using γ_{ST} alone to scale S→B can achieve the pretrained DeiT-S accuracy (79%) at step 0. γ_{Pad0} can achieve 73% accuracy with 200 steps of model training. This means training these linear mappings for increasing the initial accuracy is redundant. Besides, as discussed in Sec.2.3, we argue that initial accuracy alone is not the key to successful model scaling.

C. Combine TripLe with KD

As we reuse the DeiT architectures, the output has two parts: (1) the output logits of distillation head o_t and (2) the output logits of classification head o_s. Assuming the output logits of the teacher model is Z_t, the corresponding teaching label would be $y_t = \arg\max_c Z_t(c)$. When KD is applied, the hard loss is defined as Eq 7.

$$\mathcal{L}_{global}^{hardDistill} = \frac{1}{2}\mathcal{L}_{CE}(\psi(o_s), y) + \frac{1}{2}\mathcal{L}_{CE}(\psi(o_t), y_t) \quad (7)$$

ψ is the softmax function. $\mathcal{L}_{CE}$ is the cross-entropy loss. During model evaluation under KD, the prediction comes from the combination of both o_s and o_t: $\bar{y} = \arg\max_c \frac{o_s+o_t}{2}(c)$.

When we disable the knowledge distillation, we follow the official DeiT implementation[1] for training and the loss is given as Eq 8.

$$\mathcal{L}_{global} = \frac{1}{2}\mathcal{L}_{CE}(\psi(\frac{o_s + o_t}{2}), y) \quad (8)$$

D. Learning Curve of NAS

For each trial, both TripLe-NAS and multi-trial NAS conduct 30 epochs of training. The learning curve of the agent during the searching phase is given in Figure 1. Generally, both multi-trial and TripLe-NAS gradually increases reward over time. The learning curve of TripLe-NAS is more stable compared to multi-trial NAS.

[1] https://github.com/facebookresearch/deit

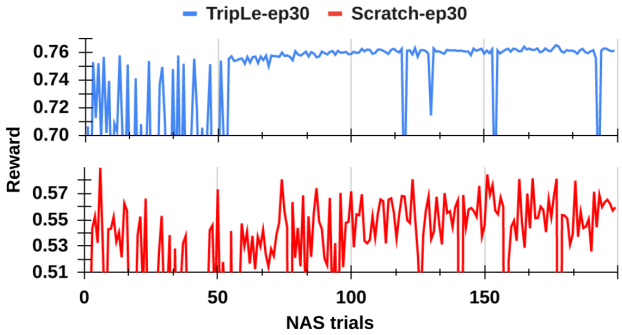

Figure 1. Learning Curve of the agents during NAS when each sample is trained with (1) TripLe$_{ep30}$ (2) Scratch$_{ep30}$.

Table 2. Transfer learning results on various datasets.

Model	Params	FLOPs	CF-10	CF-100	Cars	Flowers
DeiT-B (official)	86M	33.7B	99.1	90.8	92.1	98.4
S→B, LTG	86M	33.7B	99.1	90.7	92.1	97.8
S→B, TripLe$_{ep300}$	86M	33.7B	99.1	90.8	92.2	98.4

E. Model Transfer Learning

Table E shows the transfer learning results of ViT-TripLe and ViT-Scratch. For the downstream tasks, the inputs are resized into 224×224.

F. Searched architectures.

Table F shows the models searched using NAS with TripLe and traditional multi-trial NAS.

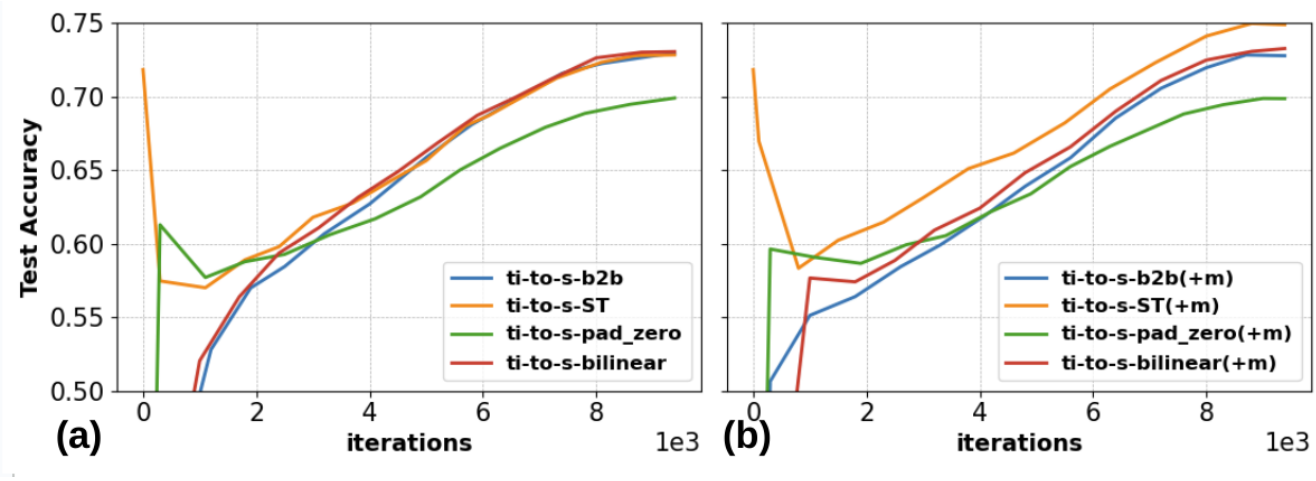

Figure 2. Training Ti→S with 30 epochs using different width expansion methods, i.e., γ_{b2B}, γ_{ST}, γ_{pad0}, γ_{intp}. '+m' denotes we also employ optimizer states in the pretrained model as discussed in Sec.2.3.

Table 3. Searched Architectures from (1) multi-trial NAS with TripLe and (2) traditional multi-trial NAS.

Model	Params	FLOPs	hidden dim	Layers	hf	ef	wd	lr
ViT-TripLe	27M	10416M	384	19	[32,32, 64,64,64,32,32,32,32,32,32,64] [32, 64,32,32,64,32,32]	[2,4,2,2,2,4,4,2,2,4,2,2] [4,4,2,2,2,4,2]	0.05	4e-3
ViT-Scratch	30M	11409M	384	19	[32,32,64,32,32,64,32,64,32,64,64,64] [32,32,32,32,32,32,32]	[3,4,4,2,3,4,2,3,4,4,4,2] [4,4,2,2,2,4,2]	0.05	4e-3

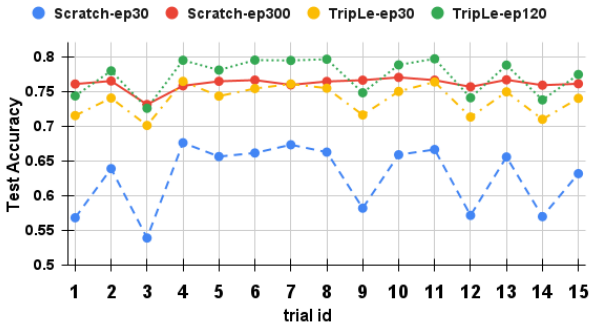

Figure 3. Task performance when trained with (1) TripLe$_{ep30}$ (2) TripLe$_{ep120}$ (3) Scratch$_{ep30}$ (4) Scratch$_{ep30}$.