Automated Progressive Learning for Efficient Training of Vision Transformers

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

Abstract

Recent advances in vision Transformers (ViTs) have come with a voracious appetite for computing power, highlighting the urgent need to develop efficient training methods for ViTs. Progressive learning, a training scheme where the model capacity grows progressively during training, has started showing its ability in efficient training. In this paper, we take a practical step towards efficient training of ViTs by customizing and automating progressive learning. First, we develop a strong manual baseline for progressive learning of ViTs, by introducing momentum growth (MoGrow) to bridge the gap brought by model growth. Then, we propose automated progressive learning (AutoProg), an efficient training scheme that aims to achieve lossless acceleration by automatically increasing the training overload on-the-fly; this is achieved by adaptively deciding whether, where and how much should the model grow during progressive learning. Specifically, we first relax the optimization of the growth schedule to sub-network architecture optimization problem, then propose one-shot estimation of the sub-network performance via an elastic supernet. The searching overhead is reduced to minimal by recycling the parameters of the supernet. Extensive experiments of efficient training on ImageNet with two representative ViT models, DeiT and VOLO, demonstrate that AutoProg can accelerate ViTs training by up to 85.1% with no performance drop.[1]

1. Introduction

With powerful high model capacity and large amounts of data, Transformers have dramatically improved the performance on many tasks in computer vision (CV) [54,69]. The pioneering ViT model [21], scales the model size to 1,021 billion FLOPs, 250× larger than ResNet-50 [31]. Through pre-training on the large-scale JFT-3B dataset [86], the recently proposed ViT model, CoAtNet [15], reached *state-of-the-art* performance, with about 8× training cost of the original ViT. The rapid growth in training scale inevitably leads to higher computation cost and carbon emissions. As shown in Tab. 1, recent breakthroughs of vision Transformers have come with a voracious appetite for computing power, resulting in considerable growth of environmental costs. Thus, it becomes extremely important to make ViTs training tenable in computation and energy consumption.

In mainstream deep learning training schemes, all the network parameters participate in every training iteration. However, we empirically found that training only a small

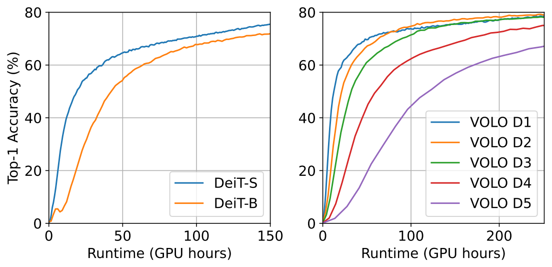

Figure 1. Accuracy of ViTs (DeiT [69] and VOLO [85]) during training. Smaller ViTs converge faster in terms of runtime[2]. Models in the legend are sorted in increasing order of model size.

Model	CO$_2$e (lbs)[3]	ImageNet Acc. (%)
ResNet-50 [21,31]	267	77.54
BERT$_{base}$ [18]	1,438	-
Avg person per year [62]	11,023	-
ViT-H/14 [21]	22,793	88.55
CoAtNet [15]	183,256	90.88

Table 1. The growth in training scale of vision models results in considerable growth of environmental costs. The CO$_2$e of human life and a language model, BERT [18] are also included for comparison. The results of ResNet-50, ViT-H/14 are from [21], and trained on JFT-300M [63]. CoAtNet is trained on JFT-3B [86].

†Corresponding author.
[1]Code: https://github.com/changlin31/AutoProg.

[2]We refer runtime to the total GPU hours used in forward and backward pass of the model during training.
[3]CO$_2$ equivalent emissions (CO$_2$e) are calculated following [56], using U.S. average energy mix, *i.e.*, 0.429 kg of CO$_2$e/KWh.

part of the parameters yields comparable performance in early training stages of ViTs. As shown in Fig. 1, smaller ViTs converge much faster in terms of runtime (though they would be eventually surpassed given enough training time). The above observation motivates us to rethink the efficiency bottlenecks of training ViTs: does every parameter, every input element need to participate in all the training steps?

Here, we make the *Growing Ticket Hypothesis* of ViTs: the performance of a large ViT model, can be reached by first training its sub-network, then the full network after properly growing, with the same total training iterations. This hypothesis generalizes the *lottery ticket hypothesis* [24] by adding a finetuning procedure at the full model size, changing its scenario from *efficient inference* to *efficient training*. By iteratively applying this hypothesis to the sub-network, we have the progressive learning scheme.

Recently, progressive learning has started showing its capability in accelerating model training. In the field of NLP, progressive learning can reduce half of BERT pre-training time [25]. Progressive learning also shows the ability to reduce the training cost for convolutional neural networks (CNNs) [66]. However, these algorithms differ substantially from each other, and their generalization ability among architectures is not well studied. For instance, we empirically observed that progressive stacking [25] could result in significant performance drop (about 1%) on ViTs.

To this end, we take a practical step towards sustainable deep learning by generalizing and automating progressive learning on ViTs. To the best of our knowledge, we are among the pioneering works to tackle the efficiency bottlenecks of training ViT models. We formulate progressive learning with its two components, *growth operator* and *growth schedule*, and study each component separately by controlling the other.

First, we present a strong manual baseline for progressive learning of ViTs by developing the growth operator. To ablate the optimization of the growth operator, we introduce a uniform linear growth schedule in two dimensions of ViTs, *i.e.*, number of patches and network depth. To bridge the gap brought by model growth, we propose *momentum growth* (**MoGrow**) operator with an effective momentum update scheme. Moreover, we present a novel *automated progressive learning* (**AutoProg**) algorithm that achieves lossless training acceleration by automatically adjusting the training overload. Specifically, we first relax the optimization of the growth schedule to sub-network architecture optimization problem. Then, we propose one-shot estimation of sub-network performance via training an *elastic supernet*. The searching overhead is reduced to minimal by recycling the parameters of the supernet.

The proposed AutoProg achieves remarkable training acceleration for ViTs on ImageNet. Without manually tuning, it consistently speeds up different ViTs training by more than 40%, on disparate variants of DeiT and VOLO, including DeiT-tiny and VOLO-D2 with 72.2% and 85.2% ImageNet accuracy, respectively. Remarkably, it accelerates VOLO-D1 [85] training by up to 85.1% with no performance drop. While significantly saving training time, AutoProg achieves competitive results when testing on larger input sizes and transferring to other datasets compared to the regular training scheme.

Overall, our contributions are as follows:

- We develop a strong manual baseline for progressive learning of ViTs, by introducing MoGrow, a momentum growth strategy to bridge the gap brought by model growing.
- We propose automated progressive learning (AutoProg), an efficient training scheme that aims to achieve lossless acceleration by automatically adjusting the growth schedule on-the-fly.
- Our AutoProg achieves remarkable training acceleration (up to 85.1%) for ViTs on ImageNet, performing favourably against the original training scheme.

2. Related Work

Progressive Learning. Early works on progressive learning [3, 23, 32, 37, 43, 59, 60, 70] mainly focus on circumventing the training difficulty of deep networks. Recently, as training costs of modern deep models are becoming formidably expensive, progressive learning starts to reveal its ability in *efficient training*. Net2Net [12] and Network Morphism [73–75] studied how to accelerate large model training by properly initializing from a smaller model. In the field of NLP, many recent works accelerate BERT pre-training by progressively stacking layers [25, 44, 79], dropping layers [88] or growing in multiple network dimensions [27]. Similar frameworks have also been proposed for efficient training of other models [71, 80]. As these algorithms remain hand-designed and could perform poorly when transferred to other networks, we propose to automate the design process of progressive learning schemes.

Automated Machine Learning. Automated Machine Learning (AutoML) aims to automate the design of model structures and learning methods from many aspects, including Neural Architecture Search (NAS) [2, 52, 64, 90], Hyper-parameter Optimization (HPO) [4, 5], AutoAugment [13, 14], AutoLoss [49, 77, 78], *etc*. By relaxing the bi-level optimization problem in AutoML, there emerges many *efficient AutoML algorithms* such as weight-sharing NAS [6, 9, 29, 45, 53, 57, 58], differentiable AutoAug [51], *etc*. These methods share network parameters in a jointly optimized *supernet* for different candidate architectures or learning methods, then rate each of these candidates according to its parameters inherited from the supernet.

Attempts have also been made on *automating progressive learning*. AutoGrow [76] proposes to use a *manually-tuned* progressive learning scheme to search for the optimal network depth, which is essentially a NAS method. LipGrow [20] could be the closest one related to our work, which adaptively decide the proper time to double the depth of CNNs on small-scale datasets, based on Lipschitz constraints. Unfortunately, LipGrow can not generalize easily to ViTs, as self-attention is not Lipschitz continuous [38]. In contrast, we solve the automated progressive learning problem from a traditional AutoML perspective, and fully automate the growing schedule by adaptively deciding *whether*, *where* and *how much* to grow. Besides, our study is conducted directly on large-scale ImageNet dataset, in accord with practical application of efficient training.

Elastic Networks. Elastic Networks, or anytime neural networks, are supernets executable with their sub-networks in various sizes, permitting instant and adaptive accuracy-efficiency trade-offs at runtime. Earlier works on Elastic Networks can be divided into *Networks with elastic depth* [34,35,41], and *networks with elastic width* [42,82,84]. The success of elastic networks is followed by their two main applications, *one-shot single-stage NAS* [8, 11, 81, 83] and *dynamic inference* [35, 47, 48, 50, 72], where emerges numerous elastic networks on *multiple dimensions* (*e.g.*, kernel size of CNNs [8, 83], head numbers [11, 33] and patch size [72] of Transformers). From a new perspective, we present an elastic Transformer serving as a sub-network performance estimator during growth for automated progressive learning.

3. Progressive Learning of Vision Transformers

In this section, we aim to develop a strong manual baseline for progressive learning of ViTs. We start by formulating progressive learning with its two main factors, *growth schedule* and *growth operator* in Sec. 3.1. Then, we present the growth space that we use in Sec. 3.2. Finally, we explore the most suitable growth operator of ViTs in Sec. 3.3.

Notations. We denote scalars, tensors and sets (or sequences) using lowercase, bold lowercase and uppercase letters (*e.g.*, n, $\boldsymbol{x}$ and Ψ). For simplicity, we use $\{\boldsymbol{x}_n\}$ to denote the set $\{\boldsymbol{x}_n\}_{n=1}^{|n|}$ with cardinality $|n|$, similarly for a sequence $(\boldsymbol{x}_n)_{n=1}^{|n|}$. Please refer to Tab. 2 for a vis-to-vis explanation of the notations we used.

3.1. Progressive Learning

Progressive learning gradually increases the training overload by growing among its sub-networks following a *growth schedule* Ψ, which can be denoted by a sequence of sub-networks with increasing sizes for all the training epochs t. In practice, to ensure the network is sufficiently optimized after each growth, it is a common practice [27,66, 79] to divide the whole training process into $|k|$ equispaced stages with $\tau = |t|/|k|$ epochs in each stage. Thus, the growth schedule can be denoted as $\Psi = \left(\boldsymbol{\psi}_k\right)_{k=1}^{|k|}$; the final one is always the complete model. Note that stages with different lengths can be achieved by using the same $\boldsymbol{\psi}$ in different numbers of consecutive stages, *e.g.*, $\Psi = (\boldsymbol{\psi}_a, \boldsymbol{\psi}_b, \boldsymbol{\psi}_b)$, where $\boldsymbol{\psi}_a, \boldsymbol{\psi}_b$ are two different sub-networks.

Notation	Type	Description
t, $\lvert t\rvert$	scalar	training epoch, total training epochs
k, $\lvert k\rvert$	scalar	training stage, total training stages
τ	scalar	epochs per stage
Ψ	sequence	growth schedule
$\boldsymbol{\zeta}$	function	growth operator
$\boldsymbol{\psi}$	network	sub-network
Φ	network	supernet
$\boldsymbol{\omega}$, $\lvert\boldsymbol{\omega}\rvert$	parameter	network parameters, number of parameters
Ω, Λ	set	the whole growth space, partial growth space
$*$, $\star$	notation	optimal, relaxed optimal

Table 2. Notations describing progressive learning and automated progressive learning.

Algorithm 1: Progressive Learning

```
Input:
$\Psi$: the growth schedule; $\boldsymbol{\zeta}$: the growth operator;
$|t|$: total training epochs; $\tau$: epochs per stage;
Randomly initialized parameters $\boldsymbol{\omega}$;
for $t \in [1, |t|]$ do
    if $t = N\tau,\ N \in \mathbb{N}_+$ then
        Switch to the sub-network of next stage $\boldsymbol{\psi} \leftarrow \Psi[t/\tau]$;
        Initialize parameters by growth operator $\boldsymbol{\omega} \leftarrow \boldsymbol{\zeta}(\boldsymbol{\omega})$;
    end
    Train $\boldsymbol{\psi}(\boldsymbol{\omega})$ over all the training data.
end
```

When growing a sub-network to a larger one, the parameters of the larger sub-network are initialized by a *growth operator* $\boldsymbol{\zeta}$, which is a reparameterization function that maps the weights $\boldsymbol{\omega}_s$ of a smaller network to $\boldsymbol{\omega}_\ell$ of a larger one by $\boldsymbol{\omega}_\ell = \boldsymbol{\zeta}(\boldsymbol{\omega}_s)$. The whole progress of progressive learning is summarized in Algorithm 1.

Let $\mathcal{L}$ be the target loss function, and $\mathcal{T}$ be the total runtime; then progressive learning can be formulated as:

$$\min_{\boldsymbol{\omega},\Psi,\boldsymbol{\zeta}} \left\{\mathcal{L}(\boldsymbol{\omega}, \Psi, \boldsymbol{\zeta}), \mathcal{T}(\Psi)\right\}, \tag{1}$$

where $\boldsymbol{\omega}$ denotes the parameters of sampled sub-networks during the optimization. Growth schedule Ψ and growth operator $\boldsymbol{\zeta}$ have been explored for language Transformers [25, 27]. However, ViTs differ considerably from their linguistic counterparts. The huge difference on task objective, data distribution and network architecture could lead to drastic difference in optimal Ψ and $\boldsymbol{\zeta}$. In the following parts of this section, we mainly study the growth operator $\boldsymbol{\zeta}$ for ViTs by fixing Ψ as a *uniform linear schedule* in a *growth space* Ω, and leave automatic exploration of Ψ to Sec. 4.

3.2. Growth Space in Vision Transformers

The model capacity of ViTs are controlled by many factors, such as number of patches, network depth, embedding dimensions, MLP ratio, *etc*. In analogy to previous discoveries on fast compound model scaling [19], we empirically find that reducing network width (*e.g.*, embedding dimen-

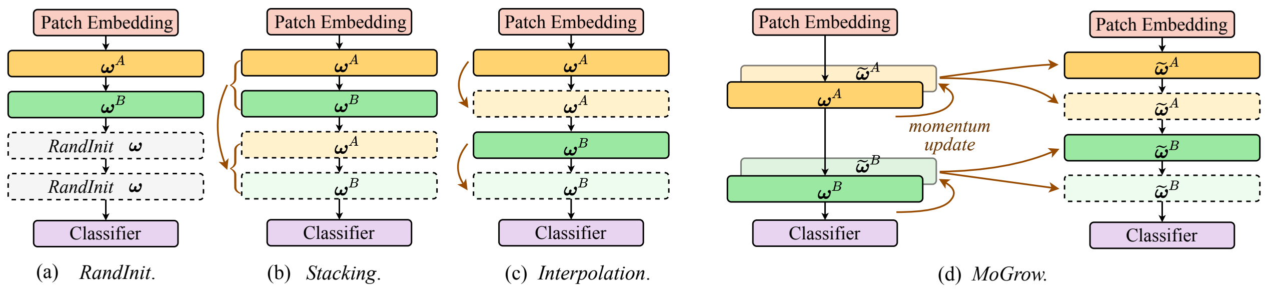

Figure 2. Variants of the growth operator ζ. ω^A and ω^B denote the parameters of two Transformer blocks in the original small network ψ_s. (a) *RandInit* randomly initializes newly added layers; (b) *Stacking* duplicates the original layers and directly stacks the duplicated ones on top of them; (c) *Interpolation* interpolate new layers of ψ_ℓ in between original ones and copy the weights from their nearest neighbor in ψ_s. (d) Our proposed *MoGrow* is build upon *Interpolation*, by coping parameters $\widetilde{\omega}$ from the momentum updated ensemble of ψ_s.

sions) yields relatively smaller wall-time acceleration on modern GPUs when comparing at the same *flops*. Thus, we mainly study *number of patchs* (n^2) and *network depth* (l), leaving other dimensions for future works.

Number of Patches. Given patch size $p \times p$, input size $r \times r$, the number of patches $n \times n$ is determined by $n^2 = r^2/p^2$. Thus, by fixing the patch size, reducing *number of patches* can be simply achieved by down-sampling the input image. However, in ViTs, the size of positional encoding is related to n. To overcome this limitation, we adaptively interpolate the positional encoding to match with n.

Network Depth. Network depth (l) is the number of Transformer blocks or its variants (*e.g.*, Outlooker blocks [85]).

Uniform Linear Growth Schedule. To ablate the optimization of growth operator ζ, we fix growth schedule Ψ as a *uniform linear growth schedule*. To be specific, *"uniform"* means that all the dimensions (*i.e.*, n and l) are scaled by the same ratio s_t at the t-th epoch; *"linear"* means that the ratio s grows linearly from s_1 to 1. This manual schedule has only one hyper-parameter, the initial scaling ratio s_1, which is set to 0.5 by default. With this fixed Ψ, the optimization of progressive learning in Eq. (1) is simplified to:

$$\min_{\omega,\zeta} \mathcal{L}(\omega,\zeta), \tag{2}$$

which enables direct optimization of ζ by comparing the final accuracy after training with different ζ.

3.3. On the Growth of Vision Transformers

Fig. 2 (a)-(c) depict the main variants of the growth operator ζ that we compare, which cover choices from a wide range of the previous works, including *RandInit* [59], *Stacking* [25] and *Interpolation* [10, 20]. More formal definitions of these schemes can be found in the supplementary material. Our empirical comparison (in Sec. 5.3) shows Interpolation growth is the most suitable scheme for ViTs.

Unfortunately, growing by Interpolation changes the original function of the network. In practice, function perturbation brought by growth can result in significant performance drop, which is hardly recovered in subsequent training steps. Early works advocate for function-preserving growth operators [12, 75], which we denote by *Identity*. However, we empirically found growing by Identity greatly harms the performance on ViTs (see Sec. 5.3). Differently, we propose a growth operator, named *Momentum Growth (MoGrow)*, to bridge the gap brought by model growth.

Momentum Growth (MoGrow). In recent years, a growing number of self-supervised [26, 28, 30] and semi-supervised [40, 68] methods learn knowledge from the historical ensemble of the network. Inspired by this, we propose to transfer knowledge from a *momentum network* during growth. This momentum network has the same architecture with ψ_s and its parameters $\widetilde{\omega}_s$ are updated with the online parameters ω_s in every training step by:

$$\widetilde{\omega}_s \leftarrow m\widetilde{\omega}_s + (1-m)\omega_s, \tag{3}$$

where m is a momentum coefficient set to 0.998. As the the momentum network usually has better generalization ability and better performance during training, loading parameters from the momentum network would help the model bypass the function perturbation gap. As shown in Fig. 2 (d), MoGrow is proposed upon Interpolation growth by maintaining a momentum network, and directly copying from it when performing network growth. MoGrow operator ζ_{MoGrow} can be simply defined as:

$$\zeta_{MoGrow}(\omega_s) = \zeta_{Interpolation}(\widetilde{\omega}_s). \tag{4}$$

4. Automated Progressive Learning

In this section, we focus on optimizing the growth schedule Ψ by fixing the growth operator as ζ_{MoGrow}. We first formulate the multi-objective optimization problem of Ψ, then propose our solution, called *AutoProg*, which is introduced in detail by its two estimation steps in Sec. 4.2 and Sec. 4.3.

4.1. Problem Formulation

The problem of designing growth schedule Ψ for efficient training is a multi-objective optimization problem [16]. By fixing ζ in Eq. (1) as our proposed ζ_{MoGrow}, the objective of designing growth schedule Ψ is:

$$\min_{\omega,\Psi} \{\mathcal{L}(\omega,\Psi), \mathcal{T}(\Psi)\}. \tag{5}$$

Note that multi-objective optimization problem has a

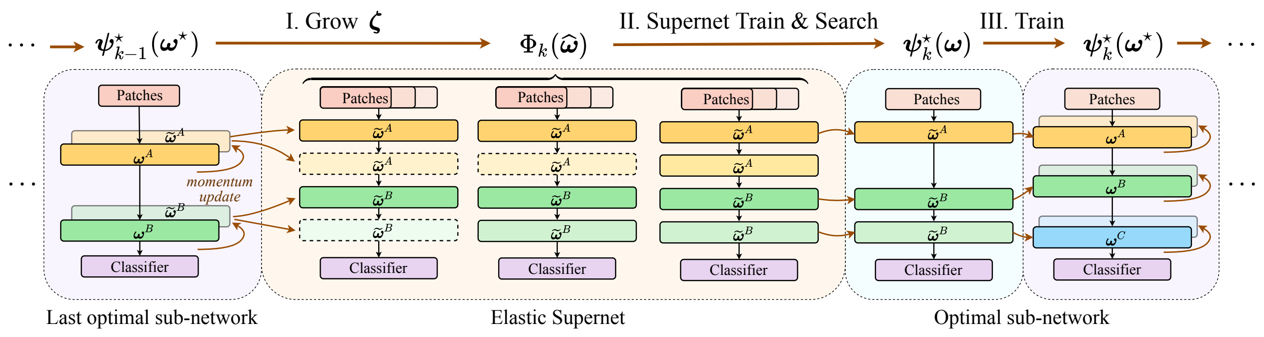

Figure 3. Pipeline of the k-th stage of automated progressive learning. In the beginning of the stage, the last optimal sub-network $\psi^\star_{k-1}$ first grows to the Elastic Supernet Φ_k by $\widehat{\omega} = \zeta(\omega^\star)$; then, we search for the optimal sub-network $\psi^\star_k$ after supernet training; finally, the sub-network is trained in the remaining epochs of this stage. The whole process of AutoProg is summarized in Algorithm 2.

Algorithm 2: Automated Progressive Learning

```
Input:
$\zeta$: the growth operator;
$|t|$: total training epochs; $\tau$: epochs per stage;
Random initialize parameters $\omega$;
for $t \in [1, |t|]$ do
    if $t = N\tau,\ N \in \mathbb{N}_+$ then
        Switch optimizers to Elastic Supernet $\Phi$;
        Initialize supernet parameters $\widehat{\omega} \leftarrow \zeta(\omega)$;
    end
    if $t = N\tau + 2,\ N \in \mathbb{N}_+$ then
        Search for the optimal sub-network $\psi$ by Eq. (9);
        Switch to the optimal sub-network $\psi \leftarrow \psi^\star$;
        Inherit parameters from the supernet $\omega \leftarrow \widehat{\omega}$;
    end
    Train $\psi(\omega)$ or supernet $\Phi(\widehat{\omega})$ over all the training data.
end
```

set of Pareto optimal [16] solutions which can be approximated using customized weighted product, a common practice used in previous Auto-ML algorithms [64, 65]. In the scenario of progressive learning, the optimization objective can be defined as:

$$\min_{\omega,\Psi} \mathcal{L}(\omega, \Psi) \cdot \mathcal{T}(\Psi)^\alpha, \tag{6}$$

where α is a balancing factor dynamically chosen by balancing the scores for all the candidate sub-networks.

4.2. Automated Progressive Learning by Optimizing Sub-Network Architectures

Denoting $|\psi|$ the number of candidate sub-networks, and $|k|$ the number of stages, the number of candidate growth schedule is thus $|\psi|^{|k|}$. As optimization of Eq. (6) contains optimization of network parameters ω, to get the final loss, a full $|t|$ epochs training with growth schedule Ψ is required:

$$\begin{aligned} \Psi^* &= \arg\min_{\Psi} \mathcal{L}\big(\omega^*(\Psi); x\big) \cdot \mathcal{T}(\Psi)^\alpha \\ \text{s.t.} \quad \omega^*(\Psi) &= \arg\min_{\omega} \mathcal{L}(\Psi, \omega; x). \end{aligned} \tag{7}$$

Thus, performing an extensive search over the higher level factor Ψ in this bi-level optimization problem has complexity $\mathcal{O}(|\psi|^{|k|} \cdot |t|)$. Its expensive cost deviates from the original intention of efficient training.

To reduce the search cost, we relax the original objective of growth schedule search to progressively deciding *whether*, *where* and *how much* should the model grow, by searching the optimal sub-network architecture ψ^*_k in each stage k. Thus, the relaxed optimal growth schedule can be denoted as $\Psi^\star = \left(\psi^*_k\right)_{k=1}^{|k|}$.

In sparse training and efficient Auto-ML algorithms, it is a common practice to estimate future ranking of models with current parameters and their gradients [22, 67], or with parameters after a single step of gradient descent update [9, 51, 53]. However, these methods are not suitable for progressive training, as the network function is drastically changed and is not stable after growth. We empirically found that the network parameters adapt quickly after growth and are already stable after one epoch of training. To make a good tradeoff between accuracy and training speed, we estimate the performance of each sub-network ψ in each stage by their training loss after the first *two* training epochs in this stage. Denoting $\omega^\star$ the sub-network parameters obtained by two epochs of training, the optimal sub-network can be searched by:

$$\begin{aligned} \psi^*_k &= \arg\min_{\psi_k \in \Lambda_k} \mathcal{L}\big(\omega^\star(\psi_k); x\big) \cdot \mathcal{T}(\psi_k)^\alpha, \\ \text{where} \quad \Lambda_k &= \Big\{ \psi \in \Omega \ \Big|\ |\omega(\psi)| \geq |\omega(\psi_k)| \Big\}, \end{aligned} \tag{8}$$

where Λ_k denotes the growth space of the k-th stage, containing all the sub-networks that are larger than or equal to the last optimal sub-network in terms of the number of parameters $|\omega|$.

Overall, by relaxing the original optimization problem in Eq. (7) to Eq. (8), we only have to train each of the $|\Lambda_k|$ sub-networks for two epochs in each of the $|k|$ stages. Thus, the search complexity is reduced exponentially from $\mathcal{O}(|\psi|^{|k|} \cdot |t|)$ to $\mathcal{O}(|\Lambda_k| \cdot |k|)$, where $|\Lambda_k| \leq |\psi|$ and $|k| \leq |t|$.

4.3. One-shot Estimation of Sub-Network Performance via Elastic Supernet

Though we relax the optimization problem with significant search cost reduction, obtaining $\omega^\star$ in Eq. (8) still takes

$2|\Lambda_k|$ epochs for each stage, which will bring huge searching overhead to the progressive learning. The inefficiency of loss prediction is caused by the repeated training of sub-networks weight $\boldsymbol{\omega}$, with bi-level optimization being its nature. To circumvent this problem, we propose to share and jointly optimize sub-network parameters in Λ_k via an *Elastic Supernet with Interpolation*.

Elastic Supernet with Interpolation. An Elastic Supernet $\Phi(\widehat{\boldsymbol{\omega}})$ is a *weight-nesting* network parameterized by $\widehat{\boldsymbol{\omega}}$, and is able to execute with its sub-networks $\{\boldsymbol{\psi}\}$. Here, we give the formal definition of *weight-nesting*:

Definition 1 *(weight-nesting) For any pair of sub-networks $\boldsymbol{\psi}_a(\boldsymbol{\omega}_a)$ and $\boldsymbol{\psi}_b(\boldsymbol{\omega}_b)$ in supernet Φ, where $|\boldsymbol{\omega}_a| \leq |\boldsymbol{\omega}_b|$, if $\boldsymbol{\omega}_a \subseteq \boldsymbol{\omega}_b$ is always satisfied, then Φ is **weight-nesting**.*

In previous works, a network with elastic depth is usually achieved by using the first layers to form sub-networks [11, 35, 83]. However, using this scheme after growing by *Interpolation* or *MoGrow* will cause inconsistency between expected sub-networks after growth and sub-networks in Φ.

To solve this issue, we present an *Elastic Supernet with Interpolation*, with optionally activated layers interpolated in between always activated ones. As shown in Fig. 3, beginning from the smaller network in the last stage $\boldsymbol{\psi}^{\star}_{k-1}$, sub-networks in Φ are formed by inserting layers in between the original layers of $\boldsymbol{\psi}^{\star}_{k-1}$ (starting from the final layers), until reaching the largest sub-network in Λ_k.

Training and Searching via Elastic Supernet. By nesting parameters of all the candidate sub-networks in the Elastic supernet Φ, the optimization of $\boldsymbol{\omega}$ is disentangled from $\boldsymbol{\psi}$. Thus, Eq. (8) is further relaxed to

$$\begin{aligned} \boldsymbol{\psi}^{\star}_k &= \underset{\boldsymbol{\psi}_k \in \Lambda_k}{\arg\min}\, \mathcal{L}\big(\widehat{\boldsymbol{\omega}}^{\star}; \boldsymbol{x}\big) \cdot \mathcal{T}(\boldsymbol{\psi}_k)^{\alpha} \\ \text{s.t.} \quad \widehat{\boldsymbol{\omega}}^{\star} &= \underset{\widehat{\boldsymbol{\omega}}}{\arg\min}\, \mathbb{E}_{\boldsymbol{\psi}_k \in \Lambda_k} \big\{\mathcal{L}(\boldsymbol{\psi}_k, \widehat{\boldsymbol{\omega}}; \boldsymbol{x})\big\}, \end{aligned} \tag{9}$$

where the optimal nested parameters $\widehat{\boldsymbol{\omega}}^{\star}$ can be obtained by one-shot training of Φ for two epochs. For efficiency, we train Φ by randomly sampling only one of its sub-networks in each step (following [11]), instead of four in [81–83].

After training all the candidate sub-networks in the Elastic Supernet Φ concurrently for two epochs, we have the adapted supernet parameters $\widehat{\boldsymbol{\omega}}^{\star}$ that can be used to estimate the real performance of the sub-networks (*i.e.* performance when trained in isolation). As the sub-network grow space Λ_k in each stage is relatively small, we can directly perform traversal search in Λ_k, by testing its training loss with a small subset of the training data. We use fixed data augmentation to ensure fair comparison, following [46]. Benefiting from parameter nesting and one-shot training of all the sub-networks in Λ_k, the search complexity is further reduced from $\mathcal{O}(|\Lambda_k| \cdot |k|)$ to $\mathcal{O}(|k|)$.

Weight Recycling. Benefiting from synergy of different sub-networks, the supernet converges at a comparable speed to training these sub-networks in isolation. Similar phenomenon can be observed in network regularization [36, 61], network augmentation [7], and previous elastic models [11, 83, 84]. Motivated by this, the searched sub-network directly inherits its parameters in the supernet to continue training. Benefiting from this *weight recycling* scheme, AutoProg has *no* extra searching epochs, since the supernet training epochs are parts of the whole training epochs. Moreover, as sampled sub-networks are faster than the full network, these supernet training epochs take less time than the original training epochs. Thus, the searching cost is directly reduced from $\mathcal{O}(|k|)$ to ***zero***.

Model	Training scheme	Speedup runtime	Top-1 (%)	Top-1@288 (%)
100 epochs				
DeiT-S [69]	Original	-	74.1	74.6
	Prog	+53.6%	72.6	73.2
	AutoProg	+40.7%	**74.4**	**74.9**
VOLO-D1 [85]	Original	-	82.6	83.0
	Prog	+60.9%	81.7	82.1
	AutoProg 0.5Ω	+65.6%	**82.8**	**83.2**
	AutoProg 0.4Ω	**+85.1%**	82.7	83.1
VOLO-D2 [85]	Original	-	83.6	84.1
	Prog	+54.4%	82.9	83.3
	AutoProg	+45.3%	**83.8**	**84.2**
300 epochs				
DeiT-Tiny [69]	Original	-	72.2	72.9
	AutoProg	+51.2%	**72.4**	**73.0**
DeiT-S [69]	Original	-	79.8	80.1
	AutoProg	+42.0%	79.8	80.1
VOLO-D1 [85]	Original	-	84.2	84.4
	AutoProg	+48.9%	**84.3**	**84.6**
VOLO-D2 [85]	Original	-	85.2	85.1
	AutoProg	+42.7%	85.2	**85.2**

Table 3. Main results of efficient training on ImageNet. Accelerations that cause accuracy drop are marked with gray. 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. Please refer to the supplementary file for detailed FLOPs and runtime.

5. Experiments

Datasets. We evaluate our method on a large scale image classification dataset, ImageNet-1K [17] and two widely used classification datasets, CIFAR-10 and CIFAR-100 [39], for transfer learning. ImageNet contains 1.2M train set images and 50K val set images in 1,000 classes. We use all the training data for progressive learning and supernet training, and use a 50K randomly sampled subset to calculate training loss for sub-network search.

Architectures. We use two representative ViT architectures, DeiT [69] and VOLO [85] to evaluate the proposed AutoProg. Specifically, DeiT [69] is a representative standard ViT model; VOLO [85] is a hybrid architecture comprised of *outlook attention* blocks and transformer blocks.

Implementation Details. For both architectures, we use

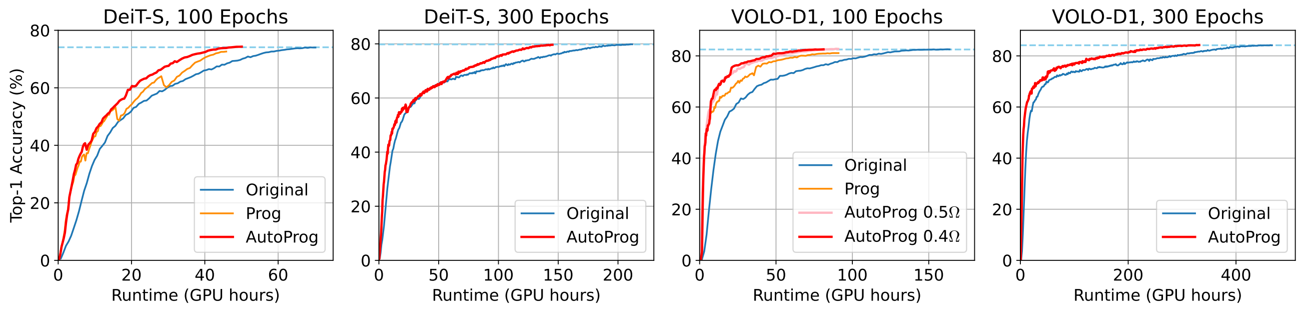

Figure 4. Evaluation accuracy of DeiT-S and VOLO-D1 during training with different learning schemes. Curves are *not* smoothed.

the original training hyper-parameters, data augmentation and regularization techniques of their corresponding prototypes [69, 85]. Our experiments are conducted on NVIDIA 3090 GPUs. As the acceleration achieved by our method is orthogonal to the acceleration of mixed precision training [55], we use it in both original baseline training and our progressive learning.

Grow Space Ω. We use 4 stages for progressive learning. The initial scaling ratio s_1 is set to 0.5 or 0.4; the corresponding grow spaces are denoted by 0.5Ω and 0.4Ω. By default, we use 0.5Ω for our experiments, unless mentioned otherwise. The grow space of n and l are calculated by multiplying the value of the whole model with 4 equispaced scaling ratios $s \in \{0.5, 0.67, 0.83, 1.0\}$, and we round the results to valid integer values. We use *Prog* to denote our manual progressive baseline with *uniform linear growth schedule* as described in Sec. 3.2.

5.1. Efficient Training on ImageNet

We first validate the effectiveness of AutoProg on ImageNet. As shown in Tab. 3, AutoProg consistently achieves remarkable efficient training results on diverse ViT architectures and training schedules.

First, our AutoProg achieves significant training acceleration over the regular training scheme with no performance drop. Generally, AutoProg speeds up ViTs training by more than 45% despite changes on training epochs and network architectures. In particular, VOLO-D1 trained with *AutoProg* 0.4Ω achieves **85.1%** training acceleration, and even slightly improves the accuracy (+0.1%). **Second**, AutoProg outperforms the manual baseline, the uniform linear growing (Prog), by a large margin. For instance, Prog scheme causes severe performance degradation on DeiT-S. AutoProg improves over Prog scheme on DeiT-S by **1.7%** on accuracy, successfully eliminating the performance gap by automatically choosing the proper growth schedule. **Third**, as progressive learning uses smaller input size during training, one may question its generalization capability on larger input sizes. We answer this by directly testing the models trained with AutoProg on 288×288 input size. The results justify that models trained with AutoProg have comparable generalization ability on larger input sizes to original models. Remarkably, VOLO-D1 trained for 300 epochs with AutoProg reaches **84.6%** Top-1 accuracy when testing on 288×288 input size, with **48.9%** faster training.

Pretrain	Speedup	CIFAR-10	CIFAR-100
Original	-	99.0	89.5
AutoProg	**48.9%**	**99.0**	**89.7**

Table 4. Transfer learning results of DeiT-S on CIFAR datasets. The evaluation metric is Top-1 accuracy (%).

The learning curves (*i.e.*, evaluation accuracy during training) of DeiT-S and VOLO-D1 with different training schemes are shown in Fig. 4. Autoprog clearly accelerates the training progress of these two models. Interestingly, DeiT-S (100 epochs) trained with manual Prog scheme presents *sharp fluctuations* after growth, while AutoProg successfully addresses this issue and eventually reaches higher accuracy by choosing proper growth schedule.

5.2. Transfer Learning

To further evaluate the transfer ability of ViTs trained with AutoProg, we conduct transfer learning on CIFAR-10 and CIFAR-100 datasets. We use the DeiT-S model that is pretrained with AutoProg on ImageNet for finetuning on CIFAR datasets, following the procedure in [69]. We compare with its counterpart pretrained with the ordinary training scheme. The results are summarized in Tab. 4. While AutoProg largely saves training time, it achieves competitive transfer learning results. This proves that AutoProg acceleration on ImageNet pretraining does not harm the transfer ability of ViTs on CIFAR datasets.

5.3. Ablation Study

Growth Operator ζ. We first compare the three growth operators mentioned in Sec. 3.3, *i.e.*, *RandInit* [59], *Stacking* [25] and *Interpolation* [10, 20], by using them with manual Prog scheme on VOLO-D1. As shown in Tab. 5, *Interpolation* growth achieves the best accuracy both after the first growth and in the final.

Then, we compare two growth operators build upon *Interpolation* scheme, our proposed MoGrow, and Identity, which is a function-preserving [12, 75] operator that can be achieved by Interpolation + ReZero [1]. Specifically,

Growth Op. ζ	Top-1@Growth (%)	Top-1 (%)
Baseline	-	82.53
RandInit [59]	60.61	80.02
Stacking [25]	61.50	81.55
Interpolation [10, 20]	**61.53**	**81.78**
Identity [12, 75]	61.04	79.32
MoGrow	**61.65**	**81.90**

Table 5. Ablation analysis of depth growth operator ζ with the Prog learning scheme. Top-1@Growth denotes the accuracy after training for the first epoch of the second stage.

Method	Top-1@Growth (%)	Top-1 (%)
AutoProg w/o MoGrow	59.41	82.6
AutoProg w/ MoGrow	**62.14**	**82.8**

Table 6. Ablation analysis of *MoGrow* in our AutoProg learning scheme on VOLO-D1. Top-1@Growth denotes the accuracy of the supernet after training for the first epoch of the second stage.

ReZero uses a zero-initialized, learnable scalar to scale the residual modules in networks. Using this technique on newly added layers can assure the original network function is preserved. The results are shown in Tab. 5. Contrary to expectations, we observe that Identity growth largely *reduces* the Top-1 accuracy of VOLO-D1 (-3.21%), probably because the network convergence is slowed down by the small scalar; besides, the global minimum of the original function could be a local minimum in the new network, which hinders the optimization. On this inferior growth schedule, our MoGrow still improves over Interpolation by 0.15%, effectively reducing its performance gap.

Previous comparisons are based on the Prog scheme. Moreover, we also analyze the effect of MoGrow on AutoProg. The results are shown in Tab. 6. We observe that MoGrow largely improves the performance of the supernet by **2.73%**. It also increases the final training accuracy by 0.2%, proving the effectiveness of MoGrow in AutoProg.

Weight Recycling. We further study the effect of weight recycling by training VOLO-D1 using AutoProg. As shown in Tab. 7, by recycling the weights of the supernet, AutoProg can achieve 12.3% more speedup. Also, benefiting from the synergy effect in weight-nesting [84], weight recycling scheme does not cause accuracy drop. These results prove the effectiveness of weight recycling.

Adaptive Regularization. Adaptive Regularization (AdaReg) for progressive learning is proposed in [66]. It adaptively change regularization intensity (including RandAug [14], Mixup [87] and Dropout [61]) according to network capacity of CNNs. Here, we generalize this scheme to ViTs and study its effect on ViT AutoProg training with DeiT-S and VOLO-D1. We mainly focus on three data augmentation and regularization techniques that are commonly used by ViTs, *i.e.*, RandAug [14], stochastic depth [36] and random erase [89]. When using AdaReg scheme, we linearly increase the magnitude of RandAug from 0.5× to 1× of its original value, and also linearly increase the probabilities of stochastic depth and random erase from 0 to their original values. The results of AutoProg with and without AdaReg are shown in Tab. 8. Notably, DeiT-S can not converge when training with AdaReg, probably because DeiT models are heavily dependent on strong augmentations. *On the contrary*, AdaReg on VOLO-D1 is *indispensable*. Not using AdaReg causes 1.2% accuracy drop on VOLO-D1. This result is consistent with previous discoveries on CNNs [66]. By default, we use AdaReg on VOLO models and not use it on DeiT models.

Method	Speedup	Top-1 Acc. (%)
w/o recycling	53.3%	82.8
w/ recycling	**65.6%**	**82.8**

Table 7. Ablation analysis of *weight recycling* in our AutoProg learning scheme on VOLO-D1.

Method	AdaReg	Speedup	Top-1 Acc. (%)
DeiT-S AutoProg	✗	**+40.7%**	**74.4**
DeiT-S AutoProg	✓	-	0.1*
VOLO-D1 AutoProg	✗	+50.9%	81.5
VOLO-D1 AutoProg	✓	**+85.1%**	**82.7**

Table 8. Ablation analysis of the adaptive regularization on ViTs with the AutoProg learning scheme. (*: training can not converge)

6. Conclusion and Discussion

In this paper, we take a practical step towards sustainable deep learning by generalizing and automating progressive learning for ViTs. We have developed a strong manual baseline for progressive learning of ViTs with MoGrow growth operator and proposed an automated progressive learning (AutoProg) scheme for automated growth schedule search. Our AutoProg has achieved consistent training speedup on different ViT models with lossless performance on ImageNet and transfer learning. Ablation studies have proved the effectiveness of each component of AutoProg.

Social Impact and Limitations. When network training becomes more efficient, it is also more available and less subject to regularization and study, which may result in a proliferation of models with harmful biases or intended uses. In this work, we achieve inspiring results with automated progressive learning on ViTs. However, large scale training of CNNs and language models can not directly benefit from it. We encourage future works to develop automated progressive learning for efficient training in broader applications.

Acknowledgement

This work was supported in part by Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) under DE190100626 and National Key R&D Program of China under Grant No. 2020AAA0109700.

References

[1] Thomas C. Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, G. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020. 7

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017. 2

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NeurIPS*, 2006. 2

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NeurIPS*, 2011. 2

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 13, 2012. 2

[6] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. In *ICLR*, 2018. 2

[7] Han Cai, Chuang Gan, Ji Lin, and Song Han. Network augmentation for tiny deep learning. *arXiv preprint arXiv:2110.08890*, 2021. 6

[8] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020. 3

[9] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 2, 5

[10] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. In *ICLR*, 2018. 4, 7, 8

[11] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *ICCV*, 2021. 3, 6

[12] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 2016. 2, 4, 7, 8

[13] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019. 2

[14] Ekin Dogus Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPRW*, 2020. 2, 8

[15] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. In *NeurIPS*, 2021. 1

[16] Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*. Springer, 2014. 4, 5

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. 1

[19] Piotr Dollár, Mannat Singh, and Ross B. Girshick. Fast and accurate model scaling. In *CVPR*, 2021. 3

[20] Chengyu Dong, Liyuan Liu, Zichao Li, and Jingbo Shang. Towards adaptive residual network training: A neural-ode perspective. In *ICML*, 2020. 3, 4, 7, 8

[21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 1

[22] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, 2020. 5

[23] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *NeurIPS*, 1989. 2

[24] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 2

[25] 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, 3, 4, 7, 8

[26] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*, 2020. 4

[27] 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, 3

[28] Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *ICML*, 2020. 4

[29] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020. 2

[30] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020. 4

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[32] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 2006. 2

[33] Lu Hou, Lifeng Shang, Xin Jiang, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. In *NeurIPS*, 2020. 3

[34] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J Andrew Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. In *AAAI*, 2019. 3

[35] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2018. 3, 6

[36] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 6, 8

[37] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018. 2

[38] Hyunjik Kim, George Papamakarios, and Andriy Mnih. The lipschitz constant of self-attention. In *ICML*, 2021. 3

[39] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009. 6

[40] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016. 4

[41] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017. 3

[42] Hankook Lee and Jinwoo Shin. Anytime neural prediction via slicing networks vertically. *arXiv preprint arXiv:1807.02609*, 2018. 3

[43] Régis Lengellé and Thierry Denoeux. Training mlps layer by layer using an objective function for internal representations. *Neural Networks*, 9, 1996. 2

[44] Bei Li, Ziyang Wang, Hui Liu, Yufan Jiang, Quan Du, Tong Xiao, Huizhen Wang, and Jingbo Zhu. Shallow-to-deep training for neural machine translation. In *EMNLP*, 2020. 2

[45] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *CVPR*, 2020. 2

[46] Changlin Li, Tao Tang, Guangrun Wang, Jiefeng Peng, Bing Wang, Xiaodan Liang, and Xiaojun Chang. Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search. In *ICCV*, 2021. 6

[47] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Ds-net++: Dynamic weight slicing for efficient inference in cnns and transformers. *arXiv preprint arXiv:2109.10060*, 2021. 3

[48] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *CVPR*, 2021. 3

[49] Hao Li, Chenxin Tao, Xizhou Zhu, Xiaogang Wang, Gao Huang, and Jifeng Dai. Auto seg-loss: Searching metric surrogates for semantic segmentation. In *ICLR*, 2021. 2

[50] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. In *ICCV*, 2019. 3

[51] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy M. Hospedales, Neil Martin Robertson, and Yongxing Yang. Dada: Differentiable automatic data augmentation. In *ECCV*, 2020. 2, 5

[52] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 2

[53] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019. 2, 5

[54] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 1

[55] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. In *ICLR*, 2018. 7

[56] David Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluís-Miquel Munguía, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021. 1

[57] Jiefeng Peng, Jiqi Zhang, Changlin Li, Guangrun Wang, Xiaodan Liang, and Liang Lin. Pi-nas: Improving neural architecture search by reducing supernet training consistency shift. In *ICCV*, 2021. 2

[58] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018. 2

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014. 2, 4, 7, 8

[60] Leslie N. Smith, Emily M. Hand, and Timothy Doster. Gradual dropin of layers to train very deep neural networks. In *CVPR*, 2016. 2

[61] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15, 2014. 6, 8

[62] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *ACL*, 2019. 1

[63] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017. 1

[64] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019. 2, 5

[65] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 5

[66] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *ICML*, 2021. 2, 3, 8

[67] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, volume 33, 2020. 5

[68] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NeurIPS*, 2017. 4

[69] 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, 6, 7

[70] Guangcong Wang, Xiaohua Xie, Jianhuang Lai, and Jiaxuan Zhuo. Deep growing learning. In *ICCV*, pages 2812–2820, 2017. 2

[71] Jiachun Wang, Fajie Yuan, Jian Chen, Qingyao Wu, Min Yang, Yang Sun, and Guoxiao Zhang. Stackrec: Efficient training of very deep sequential recommender models by iterative stacking. In *ACM SIGIR*, 2021. 2

[72] Yulin Wang, Rui Huang, Shiji Song, Zeyi Huang, and Gao Huang. Not all images are worth 16x16 words: Dynamic vision transformers with adaptive sequence length. In *NeurIPS*, 2021. 3

[73] Tao Wei, Changhu Wang, and Chang Wen Chen. Modularized morphing of neural networks. *arXiv preprint arXiv:1701.03281*, 2017. 2

[74] Tao Wei, Changhu Wang, and Chang Wen Chen. Modularized morphing of deep convolutional neural networks: A graph approach. *IEEE Transactions on Computers*, 70, 2021. 2

[75] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *ICML*, 2016. 2, 4, 7, 8

[76] Wei Wen, Feng Yan, and Hai Helen Li. Autogrow: Automatic layer growing in deep convolutional networks. In *KDD*, 2020. 2

[77] Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. Learning to teach with dynamic loss functions. In *NeurIPS*, 2018. 2

[78] Haowen Xu, H. Zhang, Zhiting Hu, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Autoloss: Learning discrete schedules for alternate optimization. In *ICLR*, 2019. 2

[79] Cheng Yang, Shengnan Wang, Chao Yang, Yuechuan Li, Ru He, and Jingqiao Zhang. Progressively stacking 2.0: A multi-stage layerwise training method for bert training speedup. *arXiv preprint arXiv:2011.13635*, 2020. 2, 3

[80] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *CVPR*, 2020. 2

[81] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. In *NeurIPS workshop*, 2019. 3, 6

[82] Jiahui Yu and Thomas S. Huang. Universally slimmable networks and improved training techniques. In *ICCV*, 2019. 3, 6

[83] 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, 6

[84] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *ICLR*, 2019. 3, 6, 8

[85] 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, 4, 6, 7

[86] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. *arXiv preprint arXiv:2106.04560*, 2021. 1

[87] Hongyi Zhang, Moustapha Cissé, Yann Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 8

[88] Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. In *NeurIPS*, 2020. 2

[89] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020. 8

[90] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2