Automated Progressive Learning for Efficient Training of Vision Transformers

Changlin Li\textsuperscript{1,2,3} Bohan Zhuang\textsuperscript{3†} Guangrun Wang\textsuperscript{4} Xiaodan Liang\textsuperscript{5} Xiaojun Chang\textsuperscript{2} Yi Yang\textsuperscript{6}

\textsuperscript{1}Baidu Research \hspace{1cm} \textsuperscript{2}ReLER, AAIL, University of Technology Sydney
\textsuperscript{3}Monash University \hspace{1cm} \textsuperscript{4}University of Oxford \hspace{1cm} \textsuperscript{5}Sun Yat-sen University \hspace{1cm} \textsuperscript{6}Zhejiang University

changlinli.ai@gmail.com, bohan.zhuang@monash.edu, wanggrun@gmail.com, xdlwang3@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. 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\times 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\times 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

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Model & $\text{CO}_2\text{e (lbs)}$ & ImageNet Acc. (%) \\
\hline
ResNet-50 [21,31] & 267 & 77.54 \\
BERT\textsubscript{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 \\
\hline
\end{tabular}
\caption{The growth in training scale of vision models results in considerable growth of environmental costs. The $\text{CO}_2\text{e}$ 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].}
\end{table}

\footnote{We refer runtime to the total GPU hours used in forward and backward pass of the model during training.}

\footnote{$\text{CO}_2\text{e}$ equivalent emissions ($\text{CO}_2\text{e}$) are calculated following [56], using U.S. average energy mix, \textit{i.e.}, 0.429 kg of CO$_2$/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 stabilize 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 recyling 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 formidable 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 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., \( n \), \( x \) and \( \Psi \)). For simplicity, we use \( \{ x_n \}_{n=1}^{|n|} \) to denote the set \( \{ x_n \}_{n=1}^{|n|} \) with cardinality \(|n|\), similarly for a sequence \( \{ x_n \}_{n=1}^{|n|} \). 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 \( \Psi \), 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 \( \Psi = \left( \psi_k \right)_{k=1}^{|k|} \); the final one is always the complete model. Note that stages with different lengths can be achieved by using the same \( \psi \) in different numbers of consecutive stages, e.g., \( \Psi = (\psi_a, \psi_b, \psi_b) \), where \( \psi_a, \psi_b \) are two different sub-networks.

When growing a sub-network to a larger one, the parameters of the larger sub-network are initialized by a growth operator \( \zeta \), which is a reparameterization function that maps the weights \( \omega_s \) of a smaller network to \( \omega_l \) of a larger one by \( \omega_l = \zeta(\omega_s) \). 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:

\[
\min_{\omega, \Psi, \zeta} \left\{ \mathcal{L}(\omega, \Psi, \zeta), \mathcal{T}(\Psi) \right\},
\]

where \( \omega \) denotes the parameters of sampled sub-networks during the optimization. Growth schedule \( \Psi \) and growth operator \( \zeta \) 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 \( \Psi \) and \( \zeta \). In the following parts of this section, we mainly study the growth operator \( \zeta \) for ViTs by fixing \( \Psi \) as a uniform linear schedule in a growth space \( \Omega \), and leave automatic exploration of \( \Psi \) 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-

<table>
<thead>
<tr>
<th>Notation</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t ), (</td>
<td>t</td>
<td>)</td>
</tr>
<tr>
<td>( k ), (</td>
<td>k</td>
<td>)</td>
</tr>
<tr>
<td>( \tau )</td>
<td>scalar</td>
<td>epochs per stage</td>
</tr>
<tr>
<td>( \Psi )</td>
<td>sequence</td>
<td>growth schedule</td>
</tr>
<tr>
<td>( \zeta )</td>
<td>function</td>
<td>growth operator</td>
</tr>
<tr>
<td>( \psi )</td>
<td>network</td>
<td>sub-network</td>
</tr>
<tr>
<td>( \Phi )</td>
<td>network</td>
<td>supernet</td>
</tr>
<tr>
<td>( \omega ), (</td>
<td>\omega</td>
<td>)</td>
</tr>
<tr>
<td>( \Omega ), (</td>
<td>\Omega</td>
<td>)</td>
</tr>
<tr>
<td>( s ), ( \Lambda )</td>
<td>notation</td>
<td>optimal, relaxed optimal</td>
</tr>
</tbody>
</table>

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

Algorithm 1: Progressive Learning

Input:

- \( \Psi \): the growth schedule; \( \zeta \): the growth operator;
- \( |t| \): total training epochs; \( \tau \): epochs per stage;

Randomly initialized parameters \( \omega \);

for \( t \in [1, |t|] \) do

if \( t = N \tau \), \( N \in \mathbb{N}_+ \) then

- Switch to the sub-network of next stage \( \psi \leftarrow \Psi(t/\tau) \);
- Initialize parameters by growth operator \( \omega \leftarrow \zeta(\omega) \);

end

Train \( \psi(\omega) \) over all the training data.

end
The problem of designing growth schedule \( \Psi \) is: \[
\min_{\omega, \zeta} \mathcal{L}(\omega, \zeta),
\] which enables direct optimization of \( \zeta \) by comparing the final accuracy after training with different \( \zeta \).

### 3.3. On the Growth of Vision Transformers

Fig. 2 (a)-(c) depict the main variants of the growth operator \( \zeta \). \( \omega^a \) and \( \omega^b \) denote the parameters of two Transformer blocks in the original small network \( \psi_s \). (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 \( \psi_s \) between original ones and copy the weights from their nearest neighbor in \( \psi_s \). (d) Our proposed MoGrow is built upon Interpolation, by coping parameters \( \bar{\omega} \) from the momentum updated ensemble of \( \psi_s \).

### 4. Automated Progressive Learning

In this section, we focus on optimizing the growth schedule \( \Psi \) by fixing the growth operator as \( \zeta_{\text{MoGrow}} \). We first formulate the multi-objective optimization problem of \( \Psi \), 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 \( \Psi \) for efficient training is a multi-objective optimization problem [16]. By fixing \( \zeta \) in Eq. (1) as our proposed \( \zeta_{\text{MoGrow}} \), the objective of designing growth schedule \( \Psi \) is:
\[
\min_{\omega, \Psi} \left\{ \mathcal{L}(\omega, \Psi), T(\Psi) \right\}.
\]
II. Supernet Train & Search

Elastic Supernet

Optimal sub-network

Patches
Classifier
Classifier

momentum update
Classifier
Classifier

I. Grow

II. Supernet Train & Search

III. Train

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 \( \Phi_k \) by \( \hat{\omega} = \zeta(\omega^\star) \); 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;
|\( \tau \)|: total training epochs; \( \tau \): epochs per stage;
Randomize initial parameters \( \omega \),

for \( t \in [1,|\tau|] \) do

if \( t \equiv N \tau \), \( N \in \mathbb{N} \) then

Switch optimizers to Elastic Supernet \( \Phi \);
Initialize supernet parameters \( \hat{\omega} \leftarrow \zeta(\omega) \);
end

if \( t \equiv N \tau + 2 \), \( N \in \mathbb{N} \) then

Search for the optimal sub-network \( \psi \) by Eq. (9);
Switch to the optimal sub-network \( \psi^\star \);
Inherit parameters from the supernet \( \omega \leftarrow \hat{\omega} \);
end

Train \( \psi(\omega) \) or supernet \( \Phi(\hat{\omega}) \) over all the training data.

end

Thus, performing an extensive search over the higher level factor \( \Psi \) in this bi-level optimization problem has complexity \( O(|\Psi|^{|k|} \cdot |\tau|) \). 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 \textit{whether, where and how much} should the model grow, by searching the optimal sub-network architecture \( \psi^\star_k \) in each stage \( k \). Thus, the relaxed optimal growth schedule can be denoted as \( \Psi^\star = \left( \psi^\star_k \right)_{k=1}^{|k|} \).

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 \( \psi \) 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:

\[
\psi^\star_k = \arg \min_{\psi_k \in \Lambda_k} \mathcal{L}(\omega^\star(\psi_k); x) \cdot \mathcal{T}(\psi_k)^\alpha,
\]

where \( \Lambda_k = \left\{ \psi \in \Omega \mid |\omega(\psi)| \geq |\omega(\psi_k)| \right\} \), and \( \Lambda_k \) 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 \( O(|\Psi|^{|k|} \cdot |\tau|) \) to \( O(|\Lambda_k| \cdot |k|) \), where |\( \Lambda_k \)| \( \leq |\Psi| \) and |\( k \)| \( \leq |\tau| \).

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

12490
2|Λ_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 ω, 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 Φ(ω) is a weight-nesting network parameterized by ω, and is able to execute with its sub-networks \{ψ\}. Here, we give the formal definition of weight-nesting:

**Definition 1** (weight-nesting) For any pair of sub-networks ψ_a(ω_a) and ψ_b(ω_b) in supernet Φ, where |ω_a| ≤ |ω_b|, if ω_a ⊆ ω_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 ψ_{k−1}^∗, sub-networks in Φ are formed by inserting layers in between the original layers of ψ_{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 ω is disentangled from ψ. Thus, Eq. (8) is further relaxed to

\[
ψ_k^* = \arg \min_{ψ_k \in Λ_k} L(\hat{ω}^*; x) \cdot T(ψ_k)\]

s.t. \[\hat{ω}^* = \arg \min_{\hat{ω} \in Λ_k} E_{ψ_k \in Λ_k} \{L(ψ_k, \hat{ω}; x)\} \]

where the optimal nested parameters \hat{ω}^* 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 \hat{ω}^* 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 O(|Λ_k| ⋅ |k|) to 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 O(|k|) to zero.

### 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
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 mod-

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

<table>
<thead>
<tr>
<th>Growth Op. (\zeta)</th>
<th>Top-1@Growth (%)</th>
<th>Top-1 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-</td>
<td>82.53</td>
</tr>
<tr>
<td>RandInit [59]</td>
<td>60.61</td>
<td>80.02</td>
</tr>
<tr>
<td>Stacking [25]</td>
<td>61.50</td>
<td>81.55</td>
</tr>
<tr>
<td>Interpolation [10, 20]</td>
<td>61.53</td>
<td>81.78</td>
</tr>
<tr>
<td>Identity [12, 75]</td>
<td>61.04</td>
<td>79.32</td>
</tr>
<tr>
<td>MoGrow</td>
<td>61.65</td>
<td>81.90</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1@Growth (%)</th>
<th>Top-1 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoProg w/o MoGrow</td>
<td>59.41</td>
<td>82.6</td>
</tr>
<tr>
<td>AutoProg w/ MoGrow</td>
<td>62.14</td>
<td>82.8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Speedup</th>
<th>Top-1 Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o recycling</td>
<td>53.3%</td>
<td>82.8</td>
</tr>
<tr>
<td>w/ recycling</td>
<td>65.6%</td>
<td>82.8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>AdaReg</th>
<th>Speedup</th>
<th>Top-1 Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeiT-S AutoProg</td>
<td>×</td>
<td>+40.7%</td>
<td>74.4</td>
</tr>
<tr>
<td>DeiT-S AutoProg</td>
<td>✓</td>
<td>-</td>
<td>0.1*</td>
</tr>
<tr>
<td>VOLO-D1 AutoProg w/</td>
<td>×</td>
<td>+50.9%</td>
<td>81.5</td>
</tr>
<tr>
<td>Recycling</td>
<td>✓</td>
<td>+85.1%</td>
<td>82.7</td>
</tr>
</tbody>
</table>

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 lin-

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


[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.


[43] Régis Lengellé and Thierry Denoeux. Training mlps layer by layer using an objective function for internal representations. Neural Networks, 9, 1996. 2
[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
[58] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In IML, 2018. 2
[68] Anti 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 ICM, 2021. 1, 6, 7

[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


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