Deep Incubation: Training Large Models by Divide-and-Conquering

Zanlin Ni[1]* Yulin Wang[1]* Jiangwei Yu[1] Haojun Jiang[1] Yue Cao[2] Gao Huang[1,2]†
[1]Department of Automation, BNRist, Tsinghua University, Beijing, China
[2]Beijing Academy of Artificial Intelligence, Beijing, China
{nzl22, wang-yl19, yu-jw19, jhj20}@mails.tsinghua.edu.cn,
caoyue10@gmail.com, gaohuang@tsinghua.edu.cn

Abstract

Recent years have witnessed a remarkable success of large deep learning models. However, training these models is challenging due to high computational costs, painfully slow convergence, and overfitting issues. In this paper, we present Deep Incubation, a novel approach that enables the efficient and effective training of large models by dividing them into smaller sub-modules which can be trained separately and assembled seamlessly. A key challenge for implementing this idea is to ensure the compatibility of the independently trained sub-modules. To address this issue, we first introduce a global, shared meta model, which is leveraged to implicitly link all the modules together, and can be designed as an extremely small network with negligible computational overhead. Then we propose a module incubation algorithm, which trains each sub-module to replace the corresponding component of the meta model and accomplish a given learning task. Despite the simplicity, our approach effectively encourages each sub-module to be aware of its role in the target large model, such that the finally-learned sub-modules can collaborate with each other smoothly after being assembled. Empirically, our method can outperform end-to-end (E2E) training in well-established training setting and shows transferable performance gain for downstream tasks (e.g., object detection and image segmentation on COCO and ADE20K). Our code is available at https://github.com/LeapLabTHU/Deep-Incubation.

1. Introduction

Large neural networks have achieved remarkable success across various domains such as natural language understanding [37, 8], computer vision [9, 57] and reinforcement learning [41, 6]. In particular, the foundation models [7, 56] heavily rely on large deep learning models to achieve state-of-the-art performance. The research field has developed a diverse set of strategies for efficient and adaptive inference of deep models [50, 48, 21, 18, 20, 36]. However, the training of large models still remains challenging in several aspects. On infrastructure side, centralized resources with strong computational and memory capacities are often required [27, 12, 57, 13]. On optimization side, the training process tends to be unstable, difficult to converge, and vulnerable to overfitting [27, 15].

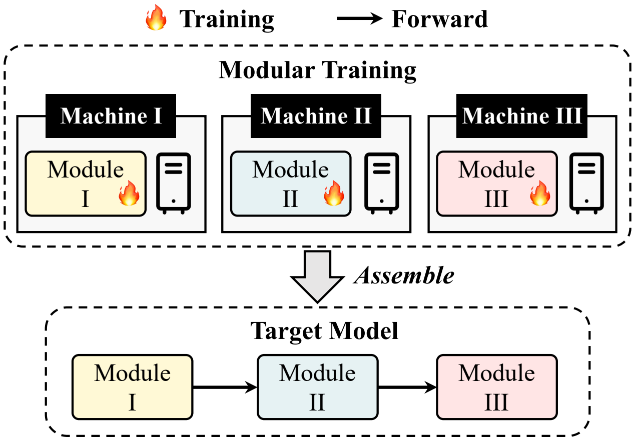


Figure 1: **An illustration of our idea**. We first train the sub-modules of a large model fully independently, and then assemble the trained modules to obtain the target model.

In this paper, we propose a *divide-and-conquer* strategy to improve the *effectiveness* (better generalization performance) and the *efficiency* (lower training cost) for training large models. In specific, we divide a large model into smaller sub-modules, train these modules separately, and then assemble them to obtain the final model. Compared with directly training the whole large network from scratch, starting the learning on top of smaller modules yields a faster and more stable converge process and higher robustness against overfitting. The independent nature also allows the training of each module to be performed on different

*Equal contribution.
†Corresponding author.

machines with no communication needed. We refer to this paradigm as "modular training", and illustrate it in Fig. 1.

Importantly, designing an effective modular training mechanism is non-trivial, as there exists a dilemma between *independency* and *compatibility*: although training sub-modules independently enjoys advantages in terms of optimization efficiency and generalization performance, it is challenging to make these modules compatible with each other when assembling them together. Some preliminary works alleviate this problem by leveraging approximated gradients [26, 11, 25] or local objectives [3, 4, 51], at the price of only achieving partial independency. However, the modules are still highly entangled during forward propagation, and generally have not exhibited the ability to effectively address the optimization issues faced by training the recently proposed large models (*e.g.*, ViTs, see Tab. 2).

In contrast, this paper proposes a Deep Incubation approach, which not only elegantly addresses this dilemma, but also demonstrates that the training of modern large models can benefit from the *divide-and-conquer* paradigm (see Tab. 1 and Fig. 4). Specifically, we first introduce a global, shared meta model, under the goal of implicitly linking all the modules together. On top of it, we propose a module incubation algorithm that trains each sub-module to replace the corresponding component of the meta model in terms of accomplishing a given learning task (*e.g.*, minimizing the supervised training loss). This design effectively encourages each sub-module to be aware of its role in the target large model. As a consequence, even though all the modules are independently trained, we are able to obtain highly compatible sub-modules which collaborate with each other smoothly after being assembled. Notably, our approach allows deploying an extremely shallow meta model, *e.g.*, only *one* layer per module, with which the computational overhead is negligible, while the performance of the target model is not affected. An overview of Deep Incubation is presented in Fig. 3.

We validate the effectiveness of Deep Incubation on the well-established DeiT training recipe [44]. Specifically, our method is able to outperform E2E training at the same training cost or deliver similar performance at a reduced training cost. Meanwhile, the performance gain is also transferable to downstream tasks like object detection on COCO [33] and semantic segmentation on ADE20K [59].

2. Related Work

Decoupled learning of neural networks is receiving more and more attention due to its biological plausibility and its potential in accelerating the model training process. Auxiliary variable methods [43, 58, 1, 31] achieve a certain level of decoupling with strong convergence guarantees. Another line of research [5, 32, 29, 35] uses biologically motivated methods to achieve decoupled learning. Using auxiliary networks [3, 4, 51] to achieve local supervision is also a way to achieve decoupling. However, most above methods focus on decoupling modules during back-propagation, while the modules are still highly entangled during forward propagation. In contrast, our modular training process completely decouples the modules and optimizes each of them independently.

Training configurations on ViTs have been extensively studied recently. Different from works that aims at improving the inference phase [49, 19], these works focus on improving the training of ViTs. The first successful training configuration on ViTs is proposed by the original ViT paper [15]. However, their configuration was effective mainly on ViTs pretrained on large datasets, *e.g.*, ImageNet-21K [14] and JFT-300M [42], while lagging behind convolutional neural networks on smaller datasets like ImageNet-1K [39]. The DeiT [44] paper proposes an improved training recipe that improves the performance of ViTs on ImageNet-1K and may be the most well-recognized configuration for training vision transformers. After the well-established DeiT training configuration, several other works further conduct more thorough hyperparameter search, architecture modifications or curriculum learning strategies to improve the supervised training of ViTs [46, 22, 45, 52, 47, 17]. In this paper, we do not opt for achieving the state-of-the-art performance on ImageNet-1K, but rather to validate the effectiveness of our method in a most well-established training setting. Hence, we adopt the DeiT training recipe as our main configuration.

Model stitching [30, 2, 10] aims to build hybrid models by "stitching" model parts from different pre-trained model with stitch layers. The aim is usually to investigate the internal representation similarity of different neural networks. A recent work [54] also applies model stitching to transfer the knowledge of pre-trained models for downstream tasks. However, the models obtained by stitching are limited by the architecture and training dataset of the pre-trained models, while our method is a general training paradigm that can be applied to any novel architectures and new datasets.

Knowledge distillation [24, 38, 40] trains a small student model to mimic the behavior of a larger model, thus transferring knowledge from the teacher model to the student model and achieves model compression. This imitative feature has some resemblance to a naïve variant of our method, which is called Module Imitation (see Fig. 2 (b)). However, they are essentially different. Specifically, the meta models in our work are much smaller than the target models, while in knowledge distillation the teacher networks are typically larger and more powerful than the student networks. Moreover, our goal is not to compress a large model into a smaller one, but to effectively train a large model with the help of a small meta model.

3. Deep Incubation

As aforementioned, training large models is typically challenging, *e.g.*, the learning process tends to be unstable, resource/data-hungry, and vulnerable to overfitting. To tackle these challenges, we propose Deep Incubation, a divide-and-conquer strategy that improves the *effectiveness* and *efficiency* of large model training. In this section, we introduce the concept of modular training. By discussing the difficulties it faces, we present our Deep Incubation approach and summarize it in Alg. 1 and Fig. 3.

Modular training first divides a large model into smaller modules, and then optimizes each module independently. As modern neural networks are generally constituted by a stack of layers, it is natural to divide the model along the depth dimension. Formally, given a large target model $\mathcal{M}$ with n layers, we can divide $\mathcal{M}$ into $K (K \leq n)$ modules:

$$\mathcal{M} = M_K \circ M_{K-1} \circ \cdots \circ M_1, \tag{1}$$

where $\circ$ represents function composition. Then, each module M_i is trained independently in modular training.

In this way, the cumbersome task of directly training a large model is decomposed into easier sub-tasks of training small modules. Moreover, these sub-tasks can be distributed to different machines and executed in full parallel, with no communication needed. After this process, we can simply assemble the trained modules, thus avoiding training the large model directly from scratch.

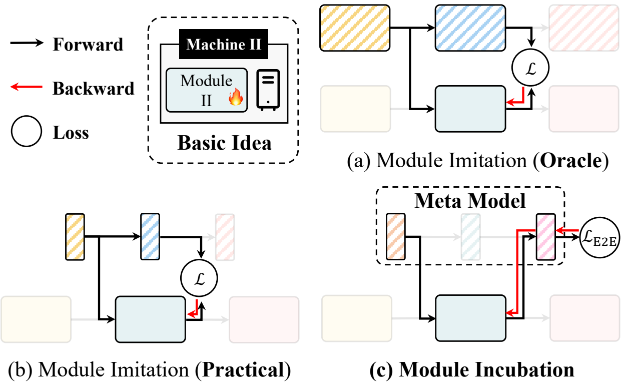
Therefore, if implemented properly, modular training can be a highly effective and efficient way for large model training. However, designing a proper modular training mechanism is a non-trivial task. In the following, we discuss in detail the challenges and present our solutions.

Dilemma I: independency *vs*. compatibility. At the core of modular training is the requirement of *independency*. However, if the modules are trained completely unaware of other modules, they may have low *compatibility* between each other, hence negatively affecting the performance of the assembled model.

Solution: meta model. We argue the root of the above dilemma is that, the requirement of independency prevents the *explicit* information exchange between modules. Consequently, the modules cannot adapt to each other during training, causing the incompatible issue. Driven by this analysis, we propose to address the dilemma by introducing a *global, shared* meta-model $\hat{\mathcal{M}}^*$ to enable *implicit* information exchange between the modules. Notably, the meta model $\hat{\mathcal{M}}^*$ is designed to have the same number of modules as the target model $\mathcal{M}$:

$$\hat{\mathcal{M}}^* = \hat{M}_K^* \circ \hat{M}_{K-1}^* \circ \cdots \circ \hat{M}_1^*, \tag{2}$$

and is pre-trained on the training dataset.



(a) Module Imitation (**Oracle**)

(b) Module Imitation (**Practical**)

(c) Module Incubation

Figure 2: **Comparison of 3 implementations of modular training** when training Module II in the target model ($K = 3$). In each implementation, the model above is the meta model $\hat{\mathcal{M}}^*$, and the model below is the target model $\mathcal{M}$. $\mathcal{L}$ is any measure of distance in feature space, *i.e.*, L_1 distance. $\mathcal{L}_{\text{E2E}}$ is the original E2E training loss. Modules not involved in the training pipeline are greyed out.

With the help of the meta model $\hat{\mathcal{M}}^*$, we can easily obtain compatible modules. For example, we can let each target module M_i imitate the behavior of meta module $\hat{M}_i^*$ by feeding it the same input as $\hat{M}_i^*$, and optimize it to produce feature similar to the output of $\hat{M}_i^*$. In this way, we can obtain compatible target modules due to the inherent compatibility between the pre-trained meta modules, thus resolving the first dilemma. We refer to this process of modular training as "Module Imitation". In an oracle case where $\hat{\mathcal{M}}^*$ has the same architecture as $\mathcal{M}$ (Fig. 2 (a)), this process can directly produce a good approximate of a well-learned target model when the trained modules are assembled.

Dilemma II: efficiency *vs*. effectiveness. Nevertheless, the solution in Fig. 2 (a) may be impractical. Since our motivation is to train $\mathcal{M}$, it is unreasonable to assume a well-learned meta model $\hat{\mathcal{M}}^*$ of the same size as $\mathcal{M}$ is already available. More importantly, adopting a large $\hat{\mathcal{M}}^*$ to facilitate modular training can incur unaffordable overhead and make the training process extremely inefficient. Therefore, in practice, a small meta model needs to be adopted for *efficiency*, as illustrated in Fig. 2 (b). However, small meta models may have insufficient representation learning ability, and thus may limit the performance of the final model. From this perspective, the meta model should not be too small for the *effectiveness* of modular training.

Solution: module incubation. We argue that the above dilemma comes from the inappropriate optimization objective for the target module M_i, which is to strictly imitate the meta module $\hat{M}_i^*$. This objective makes the representation learning ability of M_i bounded by $\hat{M}_i^*$. Therefore, we propose a "Module Incubation" mechanism to better leverage the meta model for modular training. In specific, instead of

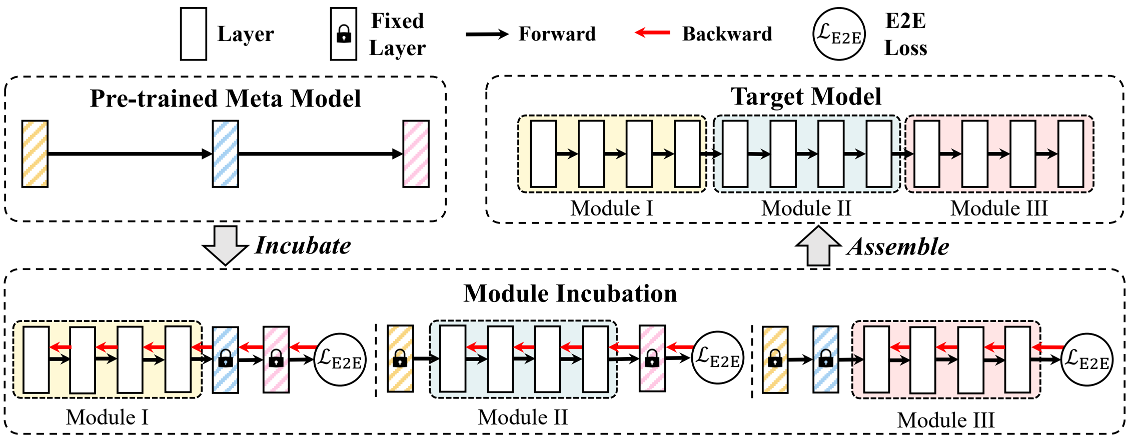


Figure 3: **The overall pipeline of Deep Incubation** ($K = 3$) **.** Here, we take a target model with 12 layers as an example, and design a meta model with only one layer per module. The meta model is end-to-end pre-trained on the training dataset. When training the i-th target module (denoted as M_i), we simply replace the i-th meta layer in the meta model with M_i, and train the resulting hybrid network in an end-to-end manner with all meta layers fixed. Then, we assemble the trained modules together to obtain the target model.

letting M_i strictly imitate $\hat{M}_i^*$, we encourage M_i to cooperate with the meta model $\hat{\mathcal{M}}^*$ to attain a task-oriented learning goal. Formally, we replace the i-th module in the meta model $\hat{\mathcal{M}}^*$ with M_i, obtaining a hybrid network $\tilde{\mathcal{M}}^{(i)}$:

$$\tilde{\mathcal{M}}^{(i)} = \hat{M}_K^* \circ \cdots \circ \hat{M}_{i+1}^* \circ M_i \circ \hat{M}_{i-1}^* \circ \cdots \circ \hat{M}_1^*. \quad (3)$$

Then we fix $\hat{M}_j^* (j \neq i)$, and thus the outputs of $\tilde{\mathcal{M}}^{(i)}$ corresponding to the input $\boldsymbol{x}$ can be defined as a function of M_i, namely:

$$\boldsymbol{x} \rightarrow \tilde{\mathcal{M}}^{(i)}(\boldsymbol{x}; M_i). \quad (4)$$

Finally, we can directly minimize an end-to-end loss $\mathcal{L}_{\text{E2E}}(\cdot)$ with respect to $\tilde{\mathcal{M}}^{(i)}(\boldsymbol{x}; M_i)$:

$$\underset{M_i}{\text{minimize}} \quad \mathcal{L}_{\text{E2E}}\left(y, \quad \tilde{\mathcal{M}}^{(i)}(\boldsymbol{x}; M_i)\right), \quad (5)$$

where y is the label of the input $\boldsymbol{x}$. Here, $\mathcal{L}_{\text{E2E}}(\cdot)$ can be defined conditioned on the task of interest. In this paper, we mainly consider the standard cross-entropy loss in the context of classification problems. The above process can be seen as using the pre-trained meta model $\hat{\mathcal{M}}^*$ to "incubate" the target module M_i, and thus we call this way of modular training "Module Incubation".

Unlike Module Imitation, here we enforce M_i to cooperate with $\hat{M}_j^* (j \neq i)$ to accomplish the final task. Therefore, M_i is encouraged to take full advantage of its potential. Since M_i is often larger than $\hat{M}_i^*$, it can acquire stronger ability than $\hat{M}_i^*$ in terms of representation learning. In contrast, the ability of M_i is generally limited by the insufficient meta module $\hat{M}_i^*$ in Module Imitation. Empirical evidence is also provided in Fig. 9 to support this point.

Interestingly, we find that smaller meta models actually bring *better* performance in Module Incubation (see Fig. 8).

Algorithm 1 The Deep Incubation Algorithm

```
Require: Initialize the target model $\mathcal{M} = M_K \circ M_{K-1} \circ \cdots \circ M_1$;
    Training dataset $\mathcal{D}$
Initialize a meta model $\hat{\mathcal{M}}$ with $K$ modules.
Pre-train $\hat{\mathcal{M}}$ on $\mathcal{D}$ to obtain $\hat{\mathcal{M}}^*$
for $i = 1$ to $K$ do                                ▷ Can be executed in parallel
    Construct $\tilde{\mathcal{M}}^{(i)}$ by replacing $\hat{M}_i^*$ in $\hat{\mathcal{M}}^*$ with $M_i$
    Minimize $\mathcal{L}_{\text{E2E}}\left(y, \tilde{\mathcal{M}}^{(i)}(\boldsymbol{x}; M_i)\right)$ on $\mathcal{D}$ to obtain $M_i^*$
end for
Assemble the target model $\mathcal{M}^{\text{assm}} = M_K^* \circ M_{K-1}^* \circ \cdots \circ M_1^*$
Fine-tune $\mathcal{M}^{\text{assm}}$ on $\mathcal{D}$ to obtain the final model $\mathcal{M}^*$
```

This intriguing phenomenon provides a favorable solution to the second dilemma, *i.e.*, we can directly use the shallowest meta model to incubate the modules. In our implementation, to get both efficiency and effectiveness, we simply design the meta model to have only *one* layer[1] per module.

Assemble the target model. After all the modules $M_i (i \in \{1, \ldots, K\})$ are trained, we obtain the target model by assembling them:

$$\mathcal{M}^{\text{assm}} = M_K^* \circ M_{K-1}^* \circ \cdots \circ M_1^*, \quad (6)$$

where M_i^* denotes the modular-trained target module. Then we fine-tune $\mathcal{M}^{\text{assm}}$ to obtain the final model $\mathcal{M}^*$. Importantly, this fine-tuning process does *not* downplay the importance of modular training. To demonstrate this issue, we can consider E2E training as a special case of Deep Incubation where the proportion of fine-tuning is 100%. However, such 100% fine-tuning significantly degrades the test accu-

[1] Following [15], we use 'layer' in a general sense to represent the basic building block of a model , *e.g.*, a transformer encoder layer in ViT.

model	image size	FLOPs	#param	E2E-ViT [15]	E2E [22]	E2E-DeiT [44]	DeiT + ours	Δ
ViT-B	224^2	17.6G	87M	-	82.3	81.8	**82.4**	**+0.6**
	384^2	55.5G		77.9	-	83.1	**84.2**	**+1.1**
ViT-L	224^2	61.6G	304M	-	82.6	$81.4^\dagger$	**83.9**	**+2.5**
	384^2	191.1G		76.5	-	$83.3^\dagger$	**85.3**	**+2.0**
ViT-H	224^2	167.4G	632M	-	83.1	$81.6^\dagger$	**84.3**	**+2.7**
	392^2	545.3G		-	-	$83.4^\dagger$	**85.6**	**+2.2**

Table 1: **Training large ViT models on ImageNet-1K**. For wall time training efficiency comparison, see Fig. 4. †: Our reproduced baseline.

dataset	model	E2E	DGL [4]	InfoPro [51]	ours
C100	ResNet-110	71.1	69.2	71.2	**73.0**
	DeiT-T-32	72.8	72.0	73.3	**75.3**
	DeiT-T-128	69.4	70.9	73.2	**77.2**
IN-1K	ViT-B	81.8	-	81.0	**82.4**

Table 2: **Comparison with decoupled learning methods.** The results on InfoPro [51] and DGL [4] are based on our implementation. C100: CIFAR-100, IN-1K: ImageNet-1K.

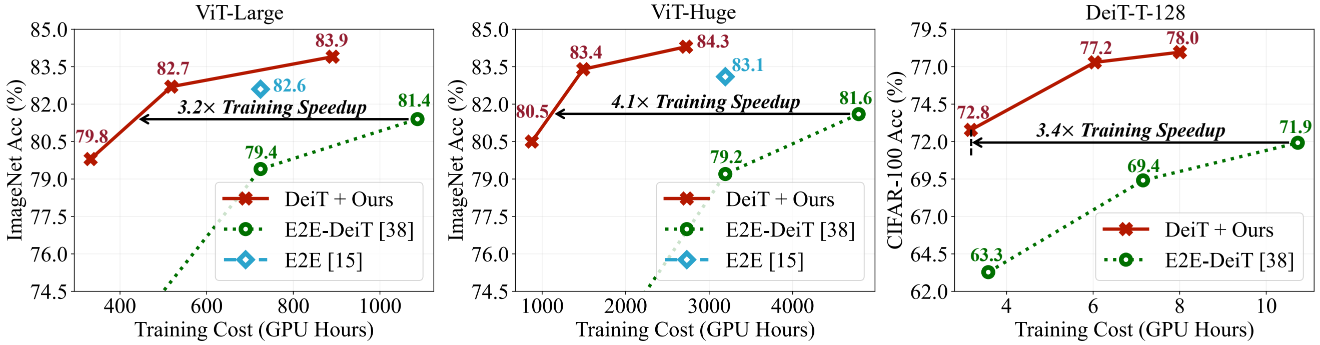


Figure 4: **Training efficiency (accuracy *vs*. training wall-time)** of ViT-L (*left*), ViT-H (*middle*) on ImageNet-1K and DeiT-T-128 on CIFAR-100 (*right*). Different points correspond to different training budgets (*i.e.*, with varying numbers of epochs). The training cost is measured in A100 GPU Hours. We also present detailed convergence curves of ViT-Huge in Fig. 5.

racy (see: Tab. 1). Only when the modular training stage is introduced, a dramatically improved generalization performance can be achieved (see: Fig. 7). This demonstrates that the major gain of our method comes from the modular training algorithm rather than the fine-tuning process.

The overall pipeline of Deep Incubation is summarized in Alg. 1 and Fig. 3.

4. Experiments

This section presents a comprehensive experimental evaluation on ImageNet-1K [39], COCO [33], ADE20K [59] and CIFAR [28] to validate the effectiveness of Deep Incubation.

Setups. We adopt the widely used training recipe of DeiT [44] as our default training configuration, with small modifications: for large models like ViT-L and ViT-H, we adapt the stochastic depth rate accordingly following [46] by setting it to 0.5 (ViT-L) and 0.6 (ViT-H) for both our fine-tuning phase and E2E baselines, and extend the warmup epochs to 20 following [34]. We additionally verify the complementarity of our method with more advanced training configurations [45], please see Appendix for more details. The E2E baselines trained with default configuration serve as our main baseline, which is denoted as E2E-DeiT. For the simplicity of our method, we intentionally adopt the *same* hyper-parameters for both our modular training and the fine-tuning phase as E2E-DeiT, except that we disable warmup in the fine-tuning phase. Therefore, we refer to our method as DeiT + Ours.

We keep $K = 4$ for modular training and evenly divide the target models. The depth of meta models is set to 4, which is the shallowest possible meta model. We simply perform modular training for 100 epochs and fine-tuning for 100 epochs. This configuration is selected for an optimal overall performance. Notably, shorter fine-tuning still produces favorable results (see Fig. 10). We pre-train meta models for 300 epochs with the same configurations as E2E-DeiT. Note that the pre-training cost of meta models is cheap compared to the overall training cost due to the shallowness of meta models (see Fig. 5). The schedules of the E2E baselines are the same as their original papers.

4.1. Main Results

Training large models on ImageNet-1K. Since the results of ViT-L and ViT-H are not reported in DeiT [46], and directly adopting the original training configurations results in optimization issues (*i.e.* NaN loss), we report our reproduced baselines. Besides the re-adjusted stochastic depth rates, we also adopt the LAMB [55] optimizer and a uniform stochastic depth rate following [46] to further improve E2E training.

As shown in Tab. 1, our method consistently improves model performance on the top of E2E-DeiT for all three ViT variants. The advantage is more pronounced for larger models. On ViT-H, our method outperforms E2E-DeiT by 2.7%.

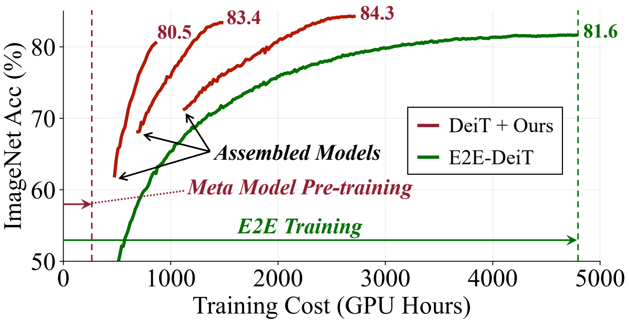


Figure 5: **Training curves of ViT-H.** Our method (with different training budgets) is compared with E2E-DeiT [44].

The advantage continues when the models are fine-tuned at higher resolution, where our method outperforms E2E-DeiT by 2.0% and 2.2% for ViT-L and ViT-H, respectively. We also compare Deep Incubation with the recently proposed improved E2E baselines in [22], where a systematically hyper-parameter search is performed on training configurations. Notably, this comparison places our method at a *disadvantage* since we directly adopt the configurations of E2E-DeiT, which may be sub-optimal for our method. Even so, Deep Incubation still performs better.

Comparison with decoupled training methods. We compare our method with two strong decoupled training methods: InfoPro [51] and DGL [4] with both ViTs and CNNs. We adopt two DeiT-Tiny [44] models with a depth of 32 and 128 and a ResNet-110 ($K = 3$) on CIFAR and ViT-B on ImageNet-1K. For models on CIFAR, we train our method for 200 (modular training) + 100 (fine-tuning) and other baselines for 400 epochs. The results are presented in Tab. 2. Our method consistently outperforms the other state-of-the-art decoupled training methods.

Higher computational efficiency for training. In Fig. 4, we present a more comprehensive comparison of the training efficiency between our method and the E2E baselines. We adjust the training cost budget by varying the number of epochs. One can observe that our method shows a better efficiency-accuracy trade-off than all E2E baselines, including the recently proposed improved E2E baselines [22].

For fair comparisons, we further discuss the benefits of our method on training efficiency by comparing it with E2E-DeiT since they adopt the same training configurations. On ViT-L and ViT-H, our method requires 3.2× and 4.1× less training cost, respectively, while achieving similar performance compared to E2E-DeiT. We also present detailed convergence curves of ViT-H in Fig. 5. For our method, we plot the convergence curve starting from the assembled models. Notably, the starting points of our convergence curves are *higher* than the convergence curve of E2E training. This demonstrates the high compatibility between the modules trained by our method.

	Mask R-CNN, 1× schedule					
	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
E2E	44.1	67.7	48.5	40.4	64.2	43.1
Ours	**45.7**(+1.6)	**69.6**(+1.9)	**50.3**(+1.8)	**41.8**(+1.4)	**66.1**(+1.9)	**44.7**(+1.6)

	Mask R-CNN, 3× schedule					
	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
E2E	47.0	69.3	51.3	42.1	66.2	45.2
Ours	**48.6**(+1.6)	**71.2**(+1.9)	**52.9**(+1.6)	**43.8**(+1.7)	**68.1**(+1.9)	**47.2**(+2.0)

Table 3: **Object detection and instance segmentation on COCO val2017**. Here, we adopt ViT-L as backbone to compare our Deep Incubation pre-training with E2E [44] and use Mask R-CNN [23] as detector. For the pre-training cost of ViT-L, see Tab. 4.

model	method	pt. cost (GPU hours)	UperNet, 80k training steps		
			mAcc.	mIoU	mIoU†
ViT-L	E2E	1.09K	58.5	47.0	47.8
	Ours	0.89K	**60.8**(+2.3)	**49.2**(+2.2)	**50.0**(+2.2)
ViT-H	E2E	4.79K	58.2	46.5	47.2
	Ours	2.72K	**61.0**(+2.8)	**49.9**(+3.4)	**50.6**(+3.4)

Table 4: **Semantic segmentation on ADE20K**. Here, we test our pre-trained models compared to the E2E trained ones [44] with UperNet [53]. † denotes the multi-scale test setting with flip augmentation.

Downstream tasks. To further demonstrate the effectiveness of our method, we evaluate our ImageNet-1K pre-trained models on 2 common downstream tasks: COCO object detection & instance segmentation and ADE20K semantic segmentation. COCO [33] object detection and instance segmentation dataset has 118K training images and 5K validation images. We employ our pre-trained models on the commonly used Mask R-CNN [23] framework with 1× and 3× training schedule. ADE20K [59] is a popular dataset for semantic segmentation with 20K training images and 2K validation images. We employ our pre-trained backbone on the widely used segmentation model UperNet [53] and train it for 80k steps. For ViT-Huge, we interpolate the patch embedding filters from 14×14×3 to 16×16×3 to fit the input image sizes of downstream tasks. As shown in Tab. 3 and 4, our pre-trained backbones achieve consistent improvement over E2E trained counterparts with a *lower* pre-training cost.

Higher data efficiency. Another important advantage of our method is its higher data efficiency, *i.e.*, it is able to dramatically outperform the E2E baselines when training data are relatively scarce. To demonstrate this, we sample two class-balanced subsets of ImageNet-1K, containing 25% and 50% of the training data, and train ViT-B on them. The training cost is kept the same as full-set training by using the same number of training iterations. Besides Top-1 accuracy, we also report the training loss in the last epoch.

training	top-1 acc.		training loss	
data	E2E-DeiT [44]	DeiT + Ours	E2E-DeiT [44]	DeiT + Ours
100% IN-1K	81.8	**82.4** (+0.6)	2.63	2.69
50% IN-1K	74.7	**78.6** (+3.9)	2.34	2.55
25% IN-1K	65.7	**72.9** (+7.2)	2.09	2.41

Table 5: **Training ViT-B with fewer training samples** (IN-1K: ImageNet-1K). Here, we sample 2 class-balanced subsets from ImageNet-1K. The training loss in the last epoch is also reported.

The results are reported in Tab. 5. It can be seen that the gain of our method is more pronounced in lower data regimes, since our method is less prone to overfitting. As data become scarce, we can observe that the *training loss* of E2E training quickly decreases, while the *validation accuracy* also drops, showing a clear trend of overfitting. In contrast, our method counters this trend with a much slower decrease in training loss, and achieves significantly higher validation accuracy than E2E training. For example, when only 25% of ImageNet-1K is available, it outperforms the DeiT baseline by **7.2%**.

4.2. Designing Deeper Models

With our proposed method, we can further explore an interesting question: *in current transformer models, is the ratio of depth* vs. *width optimal?* The answer may be true in the context of E2E learning. Previous work [16, 60] conjecture that deeper ViTs trained in an end-to-end manner may have the over-smoothing problem, hindering their performance and hence it is not suggested to make ViTs deeper than their current design.

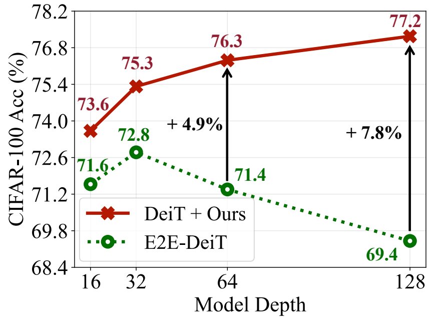


Figure 6: **Increasing depth** of DeiT-Tiny. Our method is able to train deeper ViTs without optimization issues.

To investigate this problem, we progressively increase the depth of a DeiT-Tiny in Fig. 6, and train them on CIFAR-100 to evaluate their performance. One can observe that the performance of E2E learning quickly saturates when depth increases to 32 and then starts to decrease as the model depth further increases. However, the same phenomenon does not occur with our method. The models trained by our method show no sign of saturation in performance and outperform E2E counterparts by increasingly larger margins as the model gets deeper. In other words, our method is able to train deeper ViTs more effectively.

model	FLOPs	#param	depth	width	top-1 acc.	
					E2E-DeiT [44]	DeiT + Ours
ViT-B	17.6G	87M	12	768	81.8	**82.4**
ViT-B-DN	17.7G	85M	24	540	78.7	**82.7**

Table 6: **Training deep-narrow models.** Here, a deep-narrow version of ViT-B is designed (denoted as 'ViT-B-DN') by doubling the depth of ViT-B with the FLOPs unchanged.

Intrigued by this observation, we conjecture that our proposed method may allow the designing of more efficient ViTs by further increasing the model depth. Therefore, we create a deep-narrow version of ViT-Base model (denoted ViT-B-DN) by doubling the depth and adjusting the width accordingly to keep the inference cost (*i.e.*, FLOPs) unchanged. As shown in Tab. 6, the deep-narrow version of ViT-B performs significantly worse than its original configuration when trained in an E2E manner. However, when trained by our method, the deep-narrow version actually outperforms the original one, giving an additional 0.3% improvement in the final performance. This provides an alternative solution for scaling up transformer models.

4.3. Discussion

In this section, we provide a more comprehensive analysis of our method. Unless mentioned otherwise, we use DeiT-T-128 as our target model and conduct experiments on CIFAR-100 dataset.

Ablation on modular training. We first try directly replacing modular training in our method with E2E training, and keep the fine-tuning stage unchanged (denoted as 'E2E training + tuning'). This results in a staged E2E training process that adopts a cosine annealing schedule with restart. The results are shown below:

modular training + tuning (ours)	E2E training + tuning	E2E
77.2	67.9	69.4

This indicates that the gain of our method does not come from the staged training process itself, which even underperforms the E2E baseline.

We further study the importance of modular training by varying its proportion in the whole process, while keeping the overall training cost unchanged. As shown in Fig. 7, starting from E2E training (the proportion of modular training is zero), the overall performance considerably improves as more computation is allocated to modular training. Furthermore, Deep Incubation outperforms E2E training within a wide range of the proportions, which also demonstrates its robustness.

Ablation on meta model. In our module incubation formulation, we pre-train and fix the meta model for incubat-

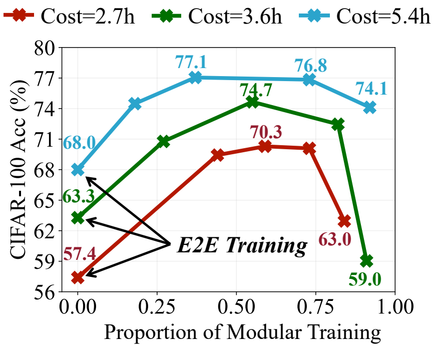


Figure 7: **Proportion of modular training**. The proportion is measured by the wall-clock time of modular training in the whole pipeline of Deep Incubation. When the proportion of modular training is zero, our method reduces to E2E training.

ing modules. The table below shows the effect of the pre-training and the fixing operation on the final performance:

pre-trained, fixed (ours)	pre-trained, tunable	random init., tunable
77.2	76.4	70.9

Thus, pre-training and fixing are both beneficial to the overall performance, with pre-training being more important. This is reasonable since the meta model is to facilitate the compatibility between independently trained modules, and thus needs to be: 1) pre-trained to ensure its own layers' compatibility and 2) fixed to be consistent when incubating different modules. Note the gain from meta model pre-training does *not* comes from the pre-trained meta model itself, which only has an accuracy of 64.9%.

We further study the effect of meta model depth in Fig. 8. The accuracy of our method is depicted in a red line, where the horizontal axis denotes the number of layers of the meta model An intriguing observation can be obtained, *i.e.*, our method achieves high accuracy even with a surprisingly shallow meta model (*e.g.*, 4 layers, one for each module). One possible explanation for this phenomenon is that, during the module incubation process, adopting shallower meta models makes the supervision information flow more easily toward the target module, and thus the target modules can be trained more thoroughly and converge faster.

Comparison with module imitation. Fig. 8 also presents the results of Module Imitation (Fig. 2 (b)), where we adopt L_1 distance as the loss function in feature space. It can be seen that our method consistently outperforms Module Imitation, especially when the meta model is small. This is aligned with our intuition in Sec. 3 that the cooperative nature of Module Incubation prevents the representation learning power of M_i from being limited by an insufficient meta model.

We also explicitly measure how well a trained target module M_i^* supports a meta model to learn representations by replacing the meta module $\hat{M}_i^*$ in the meta model with M_i^*. The accuracy gain of this hybrid model over the original meta model, which is DeiT-T-4, is evaluated. As the results in Fig. 9 show, the modules trained by Module Incubation (ours) do provide better support for the meta model by leveraging its stronger ability in representation learning.

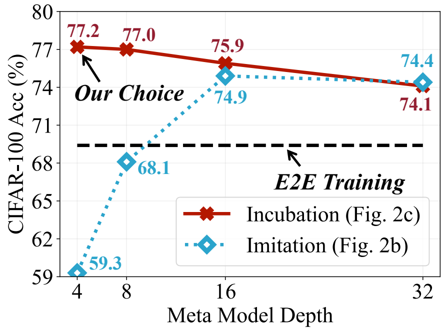


Figure 8: **Depth of the meta model**. We perform modular training with meta models of varying depths. Two ways of implementation, *i.e.*, Module Imitation (Fig. 2b) and Module Incubation (ours, Fig. 2c), are compared.

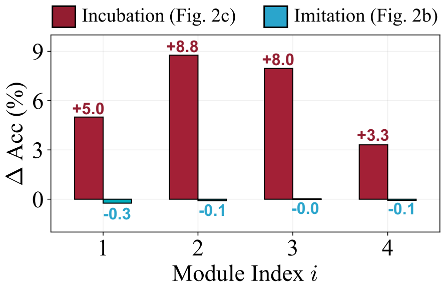


Figure 9: **Accuracy gain** when replacing a meta module $\hat{M}_i^*$ in the meta model with target module M_i^* trained by different methods.

Sensitivity test. We further conduct a sensitivity test on the hyper-parameters for fine-tuning the assembled model, namely, the epochs and the learning rate for fine-tuning. The results are shown in Fig. 10, where we use DeiT-T-128 as the target model. Three important observations can be obtained. *First*, our method can outperform E2E training even if the model is only fine-tuned for **one** epoch (71.2% for ours *vs*. 69.4% for E2E), which clearly demonstrates the necessity of our modular training process. *Second*, the majority of the performance gain can be obtained by fine-tuning the assembled model for a short period (*e.g.*, 20 epochs), and further prolonging the fine-tuning phase gives diminishing returns. *Third*, the performance of our method is generally robust to the choice of the learning rate of fine-tuning. For a moderate period of fine-tuning, directly choosing the default learning rate is enough. Therefore, for all the experiments, we do not tune this learning rate to keep the simplicity of our method.

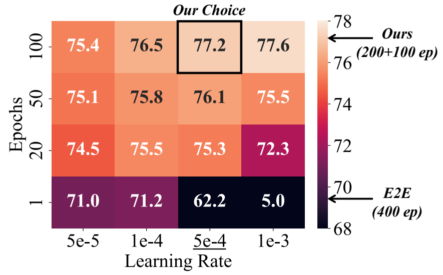


Figure 10: **Sensitivity test** on the hyper-parameters of fine-tuning on CIFAR-100. The default learning rate is underlined.

Number of modules K. Finally, we also present our study on K, which is the number of modules when we divide a target model. The results are presented below:

model	$K = 2$	$K = 4$	$K = 8$	$K = 16$	E2E
DeiT-T-32	72.3	**76.1**	75.6	75.6	72.8
DeiT-T-256	70.9	76.7	**77.2**	75.0	66.9

It can be seen that the optimal value of K differs for models of different depths, and the deeper model prefers a larger K. This is reasonable since gradient vanishing and other optimization problems get more severe for deeper models, and thus a finer division of the model is needed.

5. Conclusion

This paper presented Deep Incubation, which trains a large model in a divide-and-conquer manner. We leveraged a shared, lightweight meta model to implicitly link all modules together. By "incubating" the modules with the meta model, we effectively encouraged each module to be aware of its role in the target large model, and thus the trained modules can collaborate smoothly after they are assembled. Empirically, we demonstrated that Deep Incubation can outperform end-to-end (E2E) training in a fair setting, and the performance gain is also transferable to downstream tasks.

Acknowledgement

This work is supported in part by the National Key R&D Program of China under Grant 2021ZD0140407, the National Natural Science Foundation of China under Grants 62022048 and 62276150, THU-Bosch JCML and Beijing Academy of Artificial Intelligence. The authors would like to sincerely thank Yue Zhao and Wen Wang for their generous help. We also appreciate the generous donation of computing resources by High-Flyer AI.

References

[1] Armin Askari, Geoffrey Negiar, Rajiv Sambharya, and Laurent El Ghaoui. Lifted neural networks. *arXiv preprint arXiv:1805.01532*, 2018. 2

[2] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. 2021. 2

[3] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *ICML*, 2019. 2

[4] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. In *ICML*, 2020. 2, 5, 6

[5] Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *ArXiv*, 2014. 2

[6] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 1

[7] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 1

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020. 1

[9] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020. 1

[10] Adrián Csiszárik, Péter Kőrösi-Szabó, Akos Matszangosz, Gergely Papp, and Dániel Varga. Similarity and matching of neural network representations. 2021. 2

[11] Wojciech M. Czarnecki, Grzegorz Swirszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. In *ICML*, 2017. 2

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

[13] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *ICML*, 2023. 1

[14] 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. 2

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020. 1, 2, 4, 5

[16] Chengyue Gong, Dilin Wang, Meng Li, Vikas Chandra, and Qiang Liu. Vision transformers with patch diversification. *arXiv preprint arXiv:2104.12753*, 2021. 7

[17] Dongchen Han, Xuran Pan, Yizeng Han, Shiji Song, and Gao Huang. Flatten transformer: Vision transformer using focused linear attention. In *ICCV*, 2023. 2

[18] Yizeng Han, Dongchen Han, Zeyu Liu, Yulin Wang, Xuran Pan, Yifan Pu, Chao Deng, Junlan Feng, Shiji Song, and Gao Huang. Dynamic perceiver for efficient visual recognition. In *ICCV*, 2023. 1

[19] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *TPAMI*, 2021. 2

[20] Yizeng Han, Yifan Pu, Zihang Lai, Chaofei Wang, Shiji Song, Junfeng Cao, Wenhui Huang, Chao Deng, and Gao Huang. Learning to weight samples for dynamic early-exiting networks. In *ECCV*, 2022. 1

[21] Yizeng Han, Zhihang Yuan, Yifan Pu, Chenhao Xue, Shiji Song, Guangyu Sun, and Gao Huang. Latency-aware spatial-wise dynamic networks. In *NeurIPS*, 2022. 1

[22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022. 2, 5, 6

[23] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 6

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[25] Zhouyuan Huo, Bin Gu, Heng Huang, et al. Decoupled parallel backpropagation with convergence guarantee. In *ICML*, 2018. 2

[26] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *ICML*, 2017. 2

[27] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

[28] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 5

[29] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *ECML PKDD*, 2015. 2

[30] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *CVPR*, 2015. 2

[31] Jia Li, Cong Fang, and Zhouchen Lin. Lifted proximal operator machines. In *AAAI*, 2019. 2

[32] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014. 2

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 2, 5, 6

[34] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022. 5

[35] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *NeurIPS*, 2016. 2

[36] Yifan Pu, Yiru Wang, Zhuofan Xia, Yizeng Han, Yulin Wang, Weihao Gan, Zidong Wang, Shiji Song, and Gao Huang. Adaptive rotated convolution for rotated object detection. In *ICCV*, 2023. 1

[37] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *Technical Report*, 2019. 1

[38] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015. 2

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 2, 5

[40] Chengchao Shen, Xinchao Wang, Youtan Yin, Jie Song, Sihui Luo, and Mingli Song. Progressive network grafting for few-shot knowledge distillation. In *AAAI*, 2021. 2

[41] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, 2017. 1

[42] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017. 2

[43] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *ICML*, 2016. 2

[44] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herv'e J'egou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 2, 5, 6, 7

[45] Hugo Touvron, Matthieu Cord, and Hervé Jégou. Deit iii: Revenge of the vit. In *ECCV*, 2022. 2, 5

[46] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *ICCV*, 2021. 2, 5

[47] Qiu-Feng Wang, Xin Geng, Shu-Xia Lin, Shi-Yu Xia, Lei Qi, and Ning Xu. Learngene: From open-world to your learning task. In *AAAI*, 2022. 2

[48] Yulin Wang, Zhaoxi Chen, Haojun Jiang, Shiji Song, Yizeng Han, and Gao Huang. Adaptive focus for efficient video recognition. In *ICCV*, 2021. 1

[49] Yulin Wang, Rui Huang, Shiji Song, Zeyi Huang, and Gao Huang. Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition. 2021. 2

[50] Yulin Wang, Kangchen Lv, Rui Huang, Shiji Song, Le Yang, and Gao Huang. Glance and focus: a dynamic approach to reducing spatial redundancy in image classification. 2020. 1

[51] Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. Revisiting locally supervised learning: an alternative to end-to-end training. In *ICLR*, 2021. 2, 5, 6

[52] Yulin Wang, Yang Yue, Rui Lu, Tianjiao Liu, Zhao Zhong, Shiji Song, and Gao Huang. Efficienttrain: Exploring generalized curriculum learning for training visual backbones. In *ICCV*, 2023. 2

[53] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018. 6

[54] Xingyi Yang, Zhou Daquan, Songhua Liu, Jingwen Ye, and Xinchao Wang. Deep model reassembly. In *NeurIPS*, 2022. 2

[55] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *ICLR*, 2020. 5

[56] Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan Li, et al. Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*, 2021. 1

[57] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *CVPR*, 2022. 1

[58] Ziming Zhang and Matthew Brand. Convergent block coordinate descent for training tikhonov regularized deep neural networks. In *NeurIPS*, 2017. 2

[59] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *IJCV*, 2019. 2, 5, 6

[60] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*, 2021. 7