Building Variable-Sized Models via Learngene Pool

Boyu Shi, Shiyu Xia, Xu Yang[*], Haokun Chen, Zhiqiang Kou, Xin Geng[†]
School of Computer Science and Engineering, Southeast University, Nanjing 210096, China
Key Laboratory of New Generation Artificial Intelligence Technology and Its Interdisciplinary Applications (Southeast University), Ministry of Education, China
{shiboyu, shiyu_xia, 101013120, chenhaokun, zhiqiang_kou, xgeng}@seu.edu.cn

Abstract

Recently, Stitchable Neural Networks (SN-Net) is proposed to stitch some pre-trained networks for quickly building numerous networks with different complexity and performance trade-offs. In this way, the burdens of designing or training the variable-sized networks, which can be used in application scenarios with diverse resource constraints, are alleviated. However, SN-Net still faces a few challenges. 1) Stitching from multiple independently pre-trained anchors introduces high storage resource consumption. 2) SN-Net faces challenges to build smaller models for low resource constraints. 3). SN-Net uses an unlearned initialization method for stitch layers, limiting the final performance. To overcome these challenges, motivated by the recently proposed Learngene framework, we propose a novel method called **Learngene Pool**. Briefly, Learngene distills the critical knowledge from a large pre-trained model into a small part (termed as **learngene**) and then expands this small part into a few variable-sized models. In our proposed method, we distill one pre-trained large model into multiple small models whose network blocks are used as **learngene instances** to construct the learngene pool. Since only one large model is used, we do not need to store more large models as SN-Net and after distilling, smaller learngene instances can be created to build small models to satisfy low resource constraints. We also insert learnable transformation matrices between the instances to stitch them into variable-sized models to improve the performance of these models. Exhaustive experiments have been implemented and the results validate the effectiveness of the proposed **Learngene Pool** compared with SN-Net.

Introduction

Deep learning models (LeCun, Bengio, and Hinton 2015; Dehghani et al. 2023) have demonstrated their applicability and significance in various fields, being deployed on diverse devices like watches, smartphones, PCs, etc (Gholami et al. 2018; Howard et al. 2017). However, the diverse resource constraints of these devices lead to variations in the scale of deep learning models (Simonyan and Zisserman 2014; He et al. 2016; Dosovitskiy et al. 2020). To design models with different scales, conventional deep learning approaches typically involve manual crafting of specific model sizes for each resource constraint (Li et al. 2022b,a; Mehta and Rastegari 2021; Li et al. 2020), necessitating training from scratch (Figure 1 (a)) or compressing the huge models into smaller models (Fang et al. 2022; Zhao et al. 2022; Hameed et al. 2021; LI et al. 2023; JI et al. 2023; LI et al. 2020). However, these approaches are time-consuming and impractical for generating differently scaled models.

To address this issue, a novel method called Stitchable Neural Network (SN-Net) (Pan, Cai, and Zhuang 2023) has been proposed. Unlike the conventional approach of training individual scale-specific models, SN-Net (Pan, Cai, and Zhuang 2023) leverages pretrained models to construct variable-sized models, resulting in significant time savings. Specifically, SN-Net (Pan, Cai, and Zhuang 2023) selects pretrained varying-size models, referred to as anchors, from a model family (e.g., DeiT-Ti/S/B (Touvron et al. 2021)), and performs stitching operations among these anchors (Figure 1 (b)). The stitching is accomplished by introducing a 1×1 convolution layer, termed a stitch layer, to establish a new forward propagation path between the two adjacent anchors. By stitching varied-sized blocks of various anchors, SN-Net (Pan, Cai, and Zhuang 2023) can generate models of varying sizes.

However, SN-Net (Pan, Cai, and Zhuang 2023) still possesses several limitations. Firstly, SN-Net (Pan, Cai, and Zhuang 2023) performs stitching operations among two or more independently-trained anchors (Figure 1 (b)), resulting in substantial consumption of storage resources. Additionally, the minimum scale of the model stitched by SN-Net (Pan, Cai, and Zhuang 2023) depends on the selected smallest anchor. If the parameters of the smallest anchor are not sufficiently small, SN-Net (Pan, Cai, and Zhuang 2023) is unable to generate smaller models. For example, when stitching between DeiT-Ti/S/B (Touvron et al. 2021), the size of the stitched model is greater than or equal to that of DeiT-Tiny (5.7M) (Touvron et al. 2021). Given a smartwatch with a target resource constraint under 5M, SN-Net cannot build a model to satisfy this constraint, as shown in figure 1 (b), Finally, SN-Net computes the parameters of stitch layers by employing the least squares method from the feature maps of the anchors' outputs: $\min \|F_{I_j}(x)W - F_{I_{j+1}}(x)\|$, where $F_{I_j}(x)$ and $F_{I_{j+1}}(x)$ are the output feature maps of two anchors, W means the transformation matrix and x is

[*]Co corresponding author.
[†]Co corresponding author.

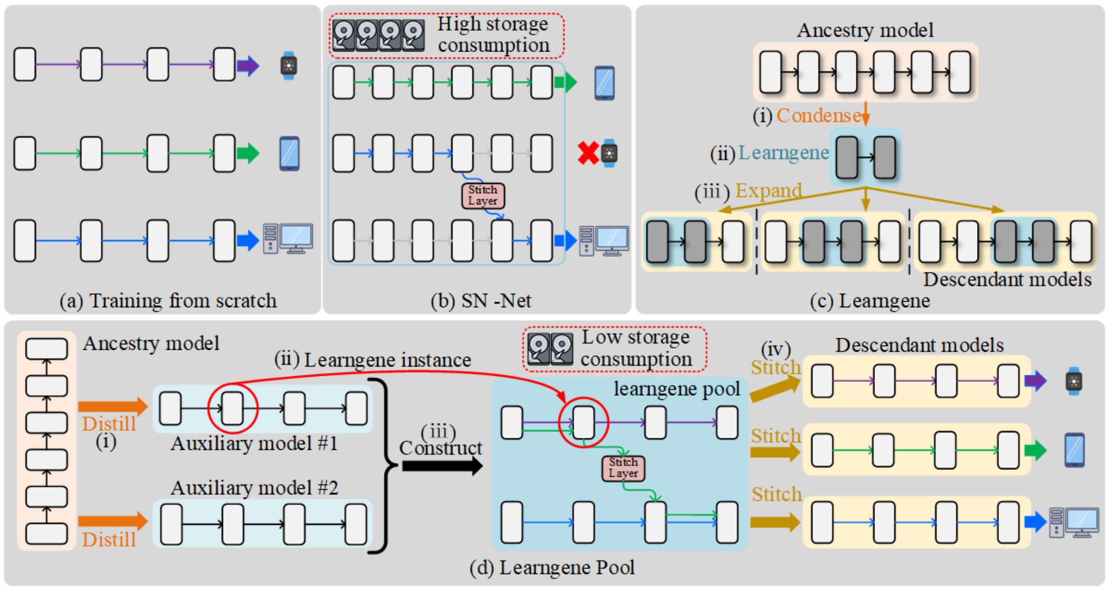

Figure 1: (a) Designing the specific model and training from scratch for all resource constraints consumes lots of resources. (b) SN-Net cannot adapt to low resource constraints and consumes lots of storage resources to save all anchors. (c) The framework of Learngene. (i& ii) Learngene condenses critical parts, termed learngene, from the large ancestry model. (iii) The extracted learngene is expanded into small or medium-sized descendant models. (d) The overall process of Learngene Pool. (i) Distilling one ancestry into multiple smaller auxiliary models. (ii& iii) Selecting each individual block in the auxiliary models as an instance to construct the learngene pool. (iv) The variable-sized descendant models are built by stitching from the learngene pool for various resource constraints.

the input samples. However, small anchors are hard to output the proper intermediate feature map, which makes this unlearnable initialization method limit the performance of the stitching layers.

One recently proposed method called Learngene (Wang et al. 2022) shows promise in mitigating the limitations of SN-Net (Pan, Cai, and Zhuang 2023). Figure 1 (c) illustrates the overall process of the Learngene. In Figure 1 (c)(i), Learngene first condenses a larger, well-trained model, termed ancestry model, into a tiny critical part known as learngene (Figure 1 (c)(ii)), which contains essential information from the ancestry model. Since the learngene is a tiny part, it consumes few storage resources. Subsequently, learngene is expanded to create many variable-sized models for downstream tasks, which are called descendant models, as shown in Figure 1 (c)(iii). Similar to SN-Net (Pan, Cai, and Zhuang 2023), Learngene utilizes well-trained models to build variable-sized models without training from scratch. Differently, Learngene generates models of different sizes by expanding the critical component, i.e., learngene, enabling the constructed models to cover smaller sizes. This effectively addresses the limitation of SN-Net (Pan, Cai, and Zhuang 2023), which faces challenges to build smaller models for low resource constraints.

However, the vanilla Learngene (Wang et al. 2022) takes a simplistic approach by extracting the last three layers of the ancestry model as learngene and combining them with randomly initialized layers to build descendant models. This approach is inadequate to fully address the challenges encountered by SN-Net (Pan, Cai, and Zhuang 2023). Inspired by the principles of Learngene, we propose a novel approach called **Learngene Pool** to comprehensively overcome the limitations of SN-Net (Pan, Cai, and Zhuang 2023).

Learngene Pool enables the construction of variable-sized models from the learngene pool, and the overall process is shown in Figure 1 (d). To establish a learngene pool, we begin by selecting a well-trained ancestry model. In this study, we adopt DeiT-Base (Touvron et al. 2021) as the ancestry model. Then, we design multiple models, referred to as the 'auxiliary models', to condense the critical knowledge of the ancestry model into smaller learngene in two ways: reducing the number of blocks and lowering the output dimensions of the blocks. In Figure 1 (d)(i), the learngene is extracted by distilling from the ancestry model to the auxiliary models. During distillation, multiple learnable transformation matrices are designed to match the output dimensions between the ancestry model and the auxiliary models.

After training, in Figure 1 (d)(ii), each block of the auxiliary models is selected as 'learngene instances' (abbreviated as the 'instance' for convenience). Subsequently, in Figure 1 (d)(iii), these selected instances collectively construct the learngene pool. Instances in it are arranged in the order of the output dimensions. Then, as shown in Figure 1 (d)(iv), we stitch learngene instances from the learngene pool to generate descendant models that meet various resource constraints. Similar to SN-Net (Pan, Cai, and Zhuang 2023), we also insert stitch layers between different learngene instances to match their output dimensions. However, we initialize the stitch layers using the parameters of the transformation matrices learned during the distillation process.

Empirically, compared to SN-Net (Pan, Cai, and Zhuang 2023), **Learngene Pool** employs a reduced number of instances, thus consuming fewer storage resources. Specifically, the learngenen pool which contains 12 and 18 instances respectively save around 59.6% and 40.1% storage resources compared to SN-Net. Furthermore, fewer instances in the learngene pool facilitate the construction of descendant models with fewer parameters. For instance, while the DeiT-based SN-Net only builds the models exceeding 5.7M parameters, the 12 and 18 instances learngenen pool can construct smaller descendant models: 3.05M and 4.38M parameters, respectively. Moreover, we initialize the stitch layers in the learngene pool by the learned block-based transformation matrices, resulting in improved final performance. Additionally, when compare **Learngene Pool** and SN-Net at the same storage resource costs, the 12 instances learngene pool improves the SN-Net results from 67.89% to 75.05% at 44.04M parameters, and the 18 instances learngene pool enhances results from 69.11% to 77.42% at 65.03M parameters.

Related Work

Learngene

The vanilla Learngene approach (Wang et al. 2022) extracts layers with stable gradients during the training of the ancestry model as learngene. Since higher-level semantic layers of the ancestry model have stable gradients, they are extracted as learngene. Then, the extracted learngene layers are combined with randomly initialized other layers to build the descendant model for the downstream tasks.

Recently, a new Learngene method (Wang et al. 2023) has been proposed, which is based on the observation (Selvaraju et al. 2020; Jiang et al. 2021) that integral layers contain critical knowledge. To extract learngene, this work first designs the pseudo descendant model for the ancestry model and trains them with the same task. Then, a meta-network is introduced to calculate the layer similarity score between the two models by following the meta-learning mechanism (Vanschoren 2018; Finn, Abbeel, and Levine 2017). The layers which have high similarity scores in the ancestry model are extracted as learngene layers. The extracted learngene layers are then stacked with various randomly initialized layers to build the descendant models.

Model Stitching

The concept of model stitching (Lenc and Vedaldi 2014) is introduced to build the new model by connecting the initial layers of one trained network with the final layers of another trained network with stitching layers. It aims to explore similarities in internal representations across different neural networks. Sequentially, (Bansal, Nakkiran, and Barak 2021; Csiszárik et al. 2021) applies model stitching to build new networks to further study the network representations. A recent work called (Yang et al. 2022) is proposed using model stitching to create a customized network for specific downstream tasks by dissecting and reassembling well-trained models.

Unlike previous approaches, SN-Net (Pan, Cai, and Zhuang 2023) is proposed to build variable-sized models by stitching from the pretrained models (termed anchors). The stitch process is achieved by inserting 1×1 convolution layers, referring to stitch layers, into these anchors. Since anchors consist of blocks of varying sizes, stitching these blocks can build various models of diverse sizes.

However, saving multiple anchors consumes lots of storage resources. Moreover, large anchors limit the minimum scale of the generated models, thus restricting their adaptability to low resource constraints. The stitch layers in SN-Net are initialized in unlearnable ways, which decreases the final performance of the built models.

Methodology

To alleviate the limitations of SN-Net, we propose a novel approach called **Learngene Pool**. This method can be divided into two main procedures: constructing the learngene pool and building the descendant model. The technical details are depicted in Figure 2. In this section, we first provide the detailed construction process of the learngene pool. Then, we illustrate the procedure for building the variable-sized descendant models.

Constructing the Learngene Pool

Selecting the Ancestry Model. To construct the learngene pool, we first carefully choose a suitable ancestry model. In our study, we adopt DeiT-Base (Touvron et al. 2021) as the ancestry model for the following reason. DeiT-Base has more parameters in the DeiT family (Touvron et al. 2021), thus granting it to learn superior representations on pretrained tasks such as ImageNet (Russakovsky et al. 2015b). This advantage enables us to extract more effective learngenes from it, which is crucial in constructing the learngene pool. The ancestry model F_{anc}^{L} with $L = 12$ blocks can be denoted as:

$$F_{anc}^{12} = f_{anc}^{H} \circ f_{anc}^{12} \circ \cdots f_{anc}^{i} \cdots \circ f_{anc}^{1} \circ f_{anc}^{PE}, \quad (1)$$

Design Auxiliary Models. To carry the critical knowledge (i.e., learngenes) extracted from the ancestry model, we design two auxiliary models for each learngene pool: the first one is to reduce the number of blocks of the ancestry model (Figure 2 (a)(i)), and the second one is to further decrease the output dimensions of each block based on the first one to obtain a smaller auxiliary model (Figure 2 (a)(ii)). For example, the output dimensions are reduced from 768 to 192. We denote the auxiliary model with l blocks as:

$$F_{aux}^{l} = f_{aux}^{H} \circ f_{aux}^{l} \circ \cdots f_{aux}^{i} \cdots \circ f_{aux}^{1} \circ f_{a}^{PE}, \quad (2)$$

where f_{aux}^{i} represents the i-th block of the auxiliary model. In this study, we conduct two experiments by designing two groups of auxiliary models (Figure 2 (a)) for validating the effectiveness of **Learngene Pool**. The first group designs auxiliary models with 6 blocks, and the second group designs auxiliary models with 9 blocks.

Training the Auxiliary Models. After designing, we extract the learngene from the ancestry model to the auxiliary

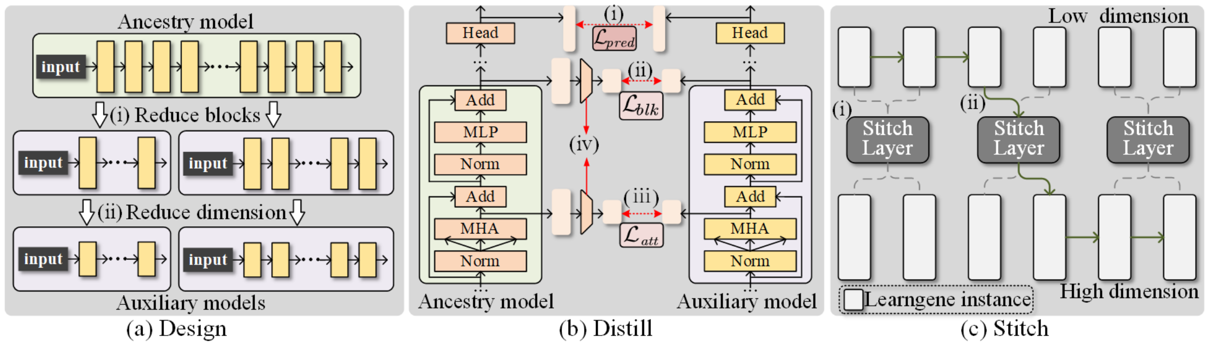

Figure 2: The technical details of the Learngene Pool. (a) The designing way of the auxiliary models. (i) Reducing the number of blocks. (ii) Reducing the output dimensions based on the ancestry model. Two groups of auxiliary models with various numbers of blocks are designed. (b) Distillation ways. (i& ii& iii) Measuring the outputs of the head operation, block, and self-attention operation between the ancestry model and the auxiliary models. (iv) Adopting the transformation matrices to match both output dimensions. (c) Stitching ways. (i) The descendant model is constructed by sequentially stitching instances in the learngene pool. (ii) Stitching from the smaller instances to the larger instances.

model by distillation. To this end, we adopt three types of distillation loss functions to transfer learngene, which is inspired by TinyBERT (Jiao et al. 2019).

Firstly, we utilize prediction-layer-based distillation (Hinton, Vinyals, and Dean 2015) to extract critical information in the prediction layer from the ancestry model into the auxiliary models. This process is achieved by adopting the soft cross-entropy loss $\mathrm{CE}_{\mathrm{soft}}(\cdot)$ between the ancestry model's output logits P_{anc} against the auxiliary models' output logits P_{aux}, and the objective is defined as:

$$\mathcal{L}_{pred} = \mathrm{CE}_{\mathrm{soft}}\left(P_{anc}/\tau, P_{aux}/\tau\right), \tag{3}$$

where τ is the temperature value of the distillation. In this study, τ has the same value as TinyBERT (Jiao et al. 2019), which is equal to 1. This process is illustrated in 2 (b)(i).

Secondly, we employ block-based distillation, aimed at transferring key knowledge contained in the blocks from the ancestry model to the blocks of the auxiliary models. This process is formulated as:

$$\mathcal{L}_{blk} = \mathrm{MSE}\left(B_{anc}W, B_{aux}\right), \tag{4}$$

where B_{anc} and B_{aux} refer to the blocks' output of the ancestry model and the auxiliary models respectively, as shown in Figure 2 (b)(ii). The matrix $W \in R^{d\times d'}$ is a learnable block-based transformation matrix, which transforms the output dimension d of B_{anc} to match the output dimension d' of B_{aux} (Figure 2 (b)(iv)).

Finally, we employ attention-based distillation, which encourages the auxiliary models to learn the informative representations of input data captured by the attention layers (Dosovitskiy et al. 2020) of the ancestry model. Specifically, as depicted in Figure 2 (b)(iii), the output of the multi-head attention layer in the auxiliary models A_{aux} is aligned with the output of the corresponding multi-head attention layer in the ancestry model A_{anc}. This function is denoted as:

$$\mathcal{L}_{att} = \mathrm{MSE}\left(A_{anc}M, A_{aux}\right), \tag{5}$$

where $M \in R^{d\times d'}$ is an attention-based transformation matrix, which transforms the output dimension d of A_{anc} into the same dimension d' as A_{aux} (Figure 2 (b)(iv)). Note that, when $d = d'$, both M and W are the simple identity matrices. Therefore, the total distillation loss is:

$$\mathcal{L}_{dis} = \mathcal{L}_{att} + \mathcal{L}_{blk} + \mathcal{L}_{pred}. \tag{6}$$

In addition to distillation, we also pre-train the auxiliary models:

$$\mathcal{L}_{cls} = \mathrm{CE}\left(y_c, F_{aux}(x)\right), \tag{7}$$

where x represents the input data and y_c denotes the label belonging to category c. Then, the total loss function of training the auxiliary models is:

$$\mathcal{L} = \alpha\mathcal{L}_{cls} + (1-\alpha)\mathcal{L}_{dis}. \tag{8}$$

Dense Distillation. We default to adopt the last block to achieve attention-based and block-based distillations. However, since the auxiliary models which output lower dimensions than the ancestry model have poor learning abilities, distilling the final block is insufficient for them to fully extract whole critical knowledge from the large ancestry model. To address this challenge, we introduce a dense distillation, which means taking multiple blocks to calculate the distillation loss functions. Specifically, we categorize both the ancestry model and auxiliary models into three levels: the low level, the middle level, and the high level based on the observation that different layers within the model exhibit varying abilities (Zeiler and Fergus 2013; Zhang, Bengio, and Singer 2019). At each level, we distill information from the final block of the ancestry model to the corresponding final block in the auxiliary models. For example, in the case of an auxiliary model with 6 blocks, we distill the 4th, 8th, and 12th blocks of the ancestry model into the 2nd, 4th, and 6th blocks of the auxiliary models. In this way, the whole critical knowledge of the ancestry model can be extracted into the auxiliary models with lower output dimensions.

After training, the critical knowledge of the ancestry model (i.e., learngene) has been extracted into all blocks of the auxiliary models. Therefore, for each auxiliary model, Algorithm 1: Building and Finetuning the Learngene Pool

```
Input: well-pretrained ancestry model $F^L_{anc}$.
Output: the learngene pool.
 1: Given $F^L_{anc}$ with output dimension $d$, narrow the number of blocks to get the learngene instance $F^{l<L}_{aux_1}$;
 2: Reduce the output dimension based on $F^{l<L}_{aux_1}$ to get $F^{l<L}_{aux_2}$ with output dimension $d' < d$;
 3: for all $k = 1$ to 2 do
 4:     for all epoch = $1, \ldots, 100$ do
 5:         Distill $F^L_{anc}$ to $F^l_{aux_k}$ and train $F^l_{aux_k}$ with Eq.8;
 6:     end for
 7: end for
 8: Select the blocks from $F^l_{aux_k}, k = 1, 2$ as learngene instances;
 9: Construct the learngene pool by learngene instances;
10: Initialize the stitch layers;
11: for all epoch = $1, \ldots, 50$ do
12:     Randomly sample one path from the learngene pool;
13:     Train the sampled path by minimizing Eq.3 and Eq.7.
14: end for
```

we select each block as one learngene instance to build the learngene pool. Within the learngene pool, learngene instances from the same auxiliary model are on a single line, arranged in order of their position in the auxiliary model. Also, the output dimension of learngene instances increases row by row in the learngen pool. Additionally, as shown in Figure 2 (c)(i), we insert multiple stitch layers between different rows in the learngene pool to transform outputs from one learngene instance to another.

Building the Descendant Models

Initialization of the Stitch Layers. SN-Net (Pan, Cai, and Zhuang 2023) calculates the parameters by the least squares method to initialize the stitch layers. However, this cannot work well when the anchor is small. To enhance the performance of the stitch layers in the learngene pool, we initialize stitch layers by the parameters of block-based transformation matrices obtained from the distillation process. Specifically, for the auxiliary models with lower output dimensions, we introduce 3 block-based transformation matrices for distilling. We average them to obtain $W \in R^{192\times768}$ and employ it to initialize all stitch layers between learngene instances with 768 dimensions and those with 192 dimensions.

Finetuning the Learngene Pool. We conduct additional training of the learngene pool to enhance its performance. The training progress takes inspiration from the work (Guo et al. 2019), where we randomly sample a single stitching path from the learngene pool and execute a single backward propagation step each time. This iterative process continues until the training reaches the end. To further improve the performance of the learngene pool, we also employ the pretrained DeiT-Base (Touvron et al. 2021) to guide the training of the learngene pool. The pipeline of building and training the learngene pool is summarized in Alg.1.

Stitching Directions. Following the establishment of the learngene pool, we proceed to build descendant models with variable sizes to satisfy diverse resource constraints. Within the learngene pool, we perform stitching operations from the smaller learngene instances to the larger ones. This operation aligns with SN-Net, demonstrating that stitching from smaller instances to larger ones yields enhanced stability and performance, as shown in Figure 2 (c)(ii).

Experiments

Implementation Setting

Dataset. We conduct all experiments on ImageNet-1K (Russakovsky et al. 2015a) dataset. ImageNet-1K is a large-scale image dataset designed for the classification task with 1,000 categories. It consists of a training set with 1.2 million images, and a validation set consisting of 50,000 images. During the training and testing phases, the initial images are resized to a resolution of 224×224.

Architectures. We adopt the DeiT-Base (Touvron et al. 2021) as the ancestry model, initialized with pre-trained parameters from Timm (Wightman 2019). Additionally, we create two auxiliary models: 6 blocks with 192 and 768 output dimensions. The two auxiliary models then construct the learngene pool which contains 12 learngene instances. For convenience, we denote it as the learngene pool (12). To further verify **Learngene Pool**, we also design larger auxiliary models: 9 blocks with 192 and 768 output dimensions. The two auxiliary models construct another learngene pool with 18 learngene instances, termed the learngene pool (18).

Training Details. We train the auxiliary models with 100 epochs and freeze the ancestry model during distillation. We employ 150 epochs for training the descendant models from scratch and 50 epochs to finetune the learngene pool. The batch size is set to 128, and the initial learning rate is set to 5×10^{-4}. All other hyperparameters remain consistent with the default setting of SN-Net (Pan, Cai, and Zhuang 2023).

Main Results and Analysis

Learngene Pool vs. SN-Net. We first conduct the comparison between the proposed **Learngene Pool** and SN-Net (Pan, Cai, and Zhuang 2023). Since SN-Net stitches from the large anchors while **Learngene Pool** employs smaller instances, it is hard to establish a one-to-one correspondence between the models constructed from these approaches. Therefore, we compare the performance of models built from the learngene pool and SN-Net under conditions of equivalent storage resource costs. Note that, we use the official code in https://github.com/ziplab/SN-Net to implement the experiments of SN-Net. The comparison results are presented in Table 1 for a resource cost of 47.83M and in Table 2 for a resource cost of 70.87M. As demonstrated, **Learngene Pool** achieves superior performance in nearly all constructed descendant models compared to SN-Net, under both resource cost scenarios.

Additionally, compared to the storage resource costs (118.4M) of SN-Net reported in (Pan, Cai, and Zhuang 2023), the learngene pool with 12 instances saves around 59.6% storage resources (118.4M vs. 47.83M), and the learngene pool with 18 instances reduces around 40.1% storage resources (118.4M vs. 70.87M). Moreover, the DeiT-

Low	High	Built Models		SN-Net	Learngene Pool (12)
		FLOPs (G)	Params (M)		
6	0	0.64	3.05	54.37	**57.00**
5	1	2.03	10.38	**65.66**	63.77
4	2	3.38	17.02	69.47	**70.00**
3	3	4.73	23.66	70.33	**72.78**
2	4	6.09	30.31	71.01	**74.21**
1	5	7.44	36.95	70.72	**74.78**
0	6	8.85	44.04	67.98	**75.05**

Table 1: The accuracy of the built models constructed by SN-Net and Learngenen Pool with 12 instances (Learngene Pool (12)). We denote the 'Low(High)' as the number of instances with low(high) output dimensions.

Low	High	Built Models		SN-Net	Learngene Pool (18)
		Flops (G)	Params (M)		
9	0	0.95	4.38	56.17	**61.63**
8	1	2.33	11.71	65.85	**66.95**
7	2	3.69	18.36	68.84	**71.69**
6	3	5.04	25.00	70.07	**74.38**
5	4	6.39	31.64	70.87	**75.69**
4	5	7.75	38.29	71.46	**76.42**
3	6	9.10	44.93	71.93	**76.95**
2	7	10.45	51.57	72.20	**77.13**
1	8	11.80	58.21	71.85	**77.32**
0	9	13.26	65.30	69.11	**77.42**

Table 2: The accuracy of the built models constructed by SN-Net and Learngenen Pool with 18 instances (Learngene Pool (18)).

based SN-Net fails to build models with parameters below 5.7M, as reported in (Pan, Cai, and Zhuang 2023). In contrast, **Learngene Pool** can construct models with 3.05M parameters from the learngene pool with 12 instances, as well as models with 4.38M parameters from the learngene pool with 18 instances, as demonstrated in the first row of Table 1 and Table 2.

Compared to training from scratch, the learngene pool with 12 instances reduces around 6.75× training costs (150+200+50 epochs vs. 18×150 epochs), and the learngene pool with 18 instances reduces around 10.13× training costs (150+200+50 epochs vs. 27×150 epochs). Note that we train the models from scratch with 150 epochs, and the saving cost can be further enlarged when training from scratch with more epochs.

12 vs. 18 instances Learngene Pools. Furthermore, we compare the performance of the descendant models built from the learngene pool with 12 and 18 instances. Noteworthy, since the sizes of descendant models built from these two learngene pools are not in one-to-one correspondence,

Built Models	Learngene Pool (12)		Learngene Pool (18)	
Params (M)	Params (M)	Acc (%)	Params (M)	Acc (%)
<= 5		57.00		**61.63**
5–15		64.30		**67.39**
15–25		73.42		**74.38**
25-35	47.83	74.54	70.87	**76.01**
35-45		75.19		**76.95**
45-55		Uncover		**77.35**
55				**77.47**

Table 3: The accuracy of the descendant models built from Learngene Pool (12) and Learngene Pool (18). 'Uncover' means the target models cannot be built.

Number	Unmatching		Matching	
	6 blocks	9 blocks	6 blocks	9 blocks
0	53.49	58.96	67.20	68.46
1	44.27	54.98	**78.83**	**80.28**
3	**53.82**	**60.44**	70.57	75.38

Table 4: The results of training the auxiliary models with various numbers of blocks used in the distillation losses. 'Matching' indicates that the output dimension of the auxiliary models matches that of the ancestry model, while 'Unmatching' means a difference. '6(9) blocks' refers to auxiliary models with 6(9) blocks.

we compare the performance of descendant models within a certain parameter size range. Within each range, we select the highest-performing descendant model. Therefore, the results, as shown in Table 3, are different from Table 1 and Table 2. We find that the descendant models tend to perform better when built from the learngene pool (18), which contains more instances in Table 3.

Ablation Studies

In this section, we ablate the number of blocks to calculate distillation loss functions when distilling the ancestry model to the auxiliary models and ways of initializing the stitch layers for finetuning the learngene pool. The training strategy is introduced in the section "Implementation Setting.".

The number of blocks to Distill. To study the effect of the number of blocks for distilling auxiliary models from the ancestry model, we consider 3 cases: 1) without the distillation, i.e., training from scratch. 2) only distilling the information from the last block in the ancestry model. 3) distilling the information of three blocks in the ancestry model, as introduced in the section "Constructing the Learngene Pool". The results are listed in Table 4.

It can be found that for auxiliary models with lower output dimensions than the ancestry model, the performance of the auxiliary models can be enhanced when incorporating three blocks to distill, as shown in the column 'Unmatch-

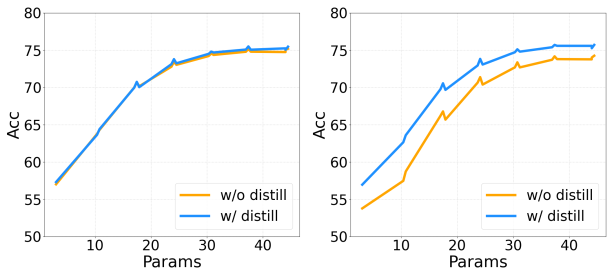

Figure 3: The performance of the descendant models built from the learngene pool which is fine-tuned with distillation and without distillation. Left: the stitch layers are initialized by the block-based transformation matrices. Right: the stitch layers are initialized by the least square method.

ing' in Table 4. We speculate that the difference in output dimensions results in the loss of critical information during distillation. Therefore, more blocks are required to fully distill information to the auxiliary model. Moreover, the performance of the auxiliary model is even lower than training from scratch when taking one block to distill. This implies that taking one block for distillation introduces a considerable amount of noise from the high-dimensional space to the low-dimensional space, resulting in a decline in the accuracy of the auxiliary models.

Conversely, for auxiliary models with identical output dimensions to the ancestry model, the distillation of individual blocks can enhance the learning of the auxiliary model, as indicated in the column 'Matching' in Table 4. This can be attributed to the fact that the same output dimension space facilitates the accurate distillation of critical information from the ancestry model to the auxiliary models, while more blocks introduce more noise.

Fine-tuning the Learngene Pool with or without distillation. To validate the application of distillation during fine-tuning the learngene pool, we compare the impact of distillation on the performance of the built descendant models, as depicted in Figure 3. We find that when the stitch layers are initialized with our block-based transformation matrices method, there is only marginal enhancement in descendant model performance, as shown in Figure 3 (Left). Initializing the stitch layers by the least square method results in a significant performance boost for the descendant models, as shown in Figure 3 (Right). This indicates that for our block-based transformation matrices method, it is unnecessary to perform distillation when fine-tuning the learngene pool. However, for the least squares method, distillation remains a crucial step during fine-tuning the learngene pool.

Initialization Ways of the Stitch Layers. We ablate the initialization ways for stitch layers in the learngene pool: the least squares method (LS) in SN-Net, and our block-based transformation matrices (TM). As shown in Figure 3 (left), the learngene pool with TM-initialized stitch layers constructs competitive descendant models even without distillation during fine-tuning. Conversely, the LS initialization results in building descendant models with inferior performances before distillation, as shown in Figure 3 (right).

Built Models		Initialization Way	
FLOPs (G)	Params (M)	LS	TM
Learngene Pool (12)			
0.64	3.05	56.95	**57.00**
2.03	10.38	62.64	**63.77**
2.13	10.82	63.59	**64.30**
3.38	17.02	69.76	**70.00**
3.48	17.47	70.55	**70.70**
Learngene Pool (18)			
0.95	4.38	60.73	**61.63**
2.33	11.71	65.07	**66.95**
2.44	12.16	65.99	**67.39**
3.79	18.80	71.63	**72.20**
5.04	25.00	73.81	**74.38**

Table 5: Accuracy comparison of descendant models built from the learngene pool with different stitch layer initialization methods: the least square methods (LS) and our block-based transformation matrices (TM).

Furthermore, We also compare the accuracy of descendant models built from the learngene pool where the stitch layers are initialized with LS and TM methods in Table 5. As it indicates, taking TM to initialize the stitch layers in the learngene pool leads to the construction of superior descendant models. These verify the effectiveness of TM for initializing stitch layers in this study.

Conclusion

In this paper, we propose a novel method called **Learngene pool** to build variable-sized descendant models for various resource constraints by inserting stitch layers among learngene instances in the learngene pool. To achieve this, we distill the larger model into multiple smaller auxiliary models. In this way, the auxiliary models can extract critical knowledge from the larger model. Then, we select all blocks in the auxiliary models as learngene instances to construct the learngene pool. The stitch layers in it are initialized by the block-based transformation matrices during the training of the auxiliary models. Since the learngene pool consists of learngene instances with varying output dimensions, stitching them results in variable-sized descendant models. Compare to SN-Net, the learngene pool consumes fewer storage resources. Moreover, the smaller learngene instances enable to build the smaller descendant models, which adapt to low resource constraints. Finally, the proposed way of initializing the stitch layers enables the learngene pool to build descendant models with superior performances.

Acknowledgments

This work was supported by the National Key Research & Development Plan of China (No. 2018AAA0100104), the National Science Foundation of China (62125602, 62076063, 62206048), Natural Science Foundation of Jiangsu Province (BK20220819), Young Elite Scientists Sponsorship Program of Jiangsu Association for Science and Technology Tj-2022-027, the Xplorer Prize, and the Big Data Computing Center of Southeast University.

References

Bansal, Y.; Nakkiran, P.; and Barak, B. 2021. Revisiting Model Stitching to Compare Neural Representations. In *Neural Information Processing Systems*.

Csiszárik, A.; Korösi-Szabó, P.; Matszangosz, Á. K.; Papp, G.; and Varga, D. 2021. Similarity and Matching of Neural Network Representations. In *Neural Information Processing Systems*.

Dehghani, M.; Djolonga, J.; Mustafa, B.; Padlewski, P.; Heek, J.; Gilmer, J.; Steiner, A.; Caron, M.; Geirhos, R.; Alabdulmohsin, I. M.; Jenatton, R.; Beyer, L.; Tschannen, M.; Arnab, A.; Wang, X.; Riquelme, C.; Minderer, M.; Puigcerver, J.; Evci, U.; Kumar, M.; van Steenkiste, S.; Elsayed, G. F.; Mahendran, A.; Yu, F.; Oliver, A.; Huot, F.; Bastings, J.; Collier, M.; Gritsenko, A. A.; Birodkar, V.; Vasconcelos, C. N.; Tay, Y.; Mensink, T.; Kolesnikov, A.; Paveti'c, F.; Tran, D.; Kipf, T.; Luvci'c, M.; Zhai, X.; Keysers, D.; Harmsen, J.; and Houlsby, N. 2023. Scaling Vision Transformers to 22 Billion Parameters. *ArXiv*, abs/2302.05442.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Fang, G.; Mo, K.; Wang, X.; Song, J.; Bei, S.; Zhang, H.; and Song, M. 2022. Up to 100x Faster Data-free Knowledge Distillation. In *AAAI Conference on Artificial Intelligence*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv: Learning,arXiv: Learning*.

Gholami, A.; Kwon, K.; Wu, B.; Tai, Z.; Yue, X.; Jin, P. H.; Zhao, S.; and Keutzer, K. 2018. SqueezeNext: Hardware-Aware Neural Network Design. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1719–171909.

Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2019. Single Path One-Shot Neural Architecture Search with Uniform Sampling. In *European Conference on Computer Vision*.

Hameed, M. G. A.; Tahaei, M. S.; Mosleh, A.; and Nia, V. 2021. Convolutional Neural Network Compression through Generalized Kronecker Product Decomposition. In *AAAI Conference on Artificial Intelligence*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hinton, G. E.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. *ArXiv*, abs/1503.02531.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv*, abs/1704.04861.

JI, Z.; NI, J.; LIU, X.; and PANG, Y. 2023. Teachers cooperation: team-knowledge distillation for multiple cross-domain few-shot learning. *Frontiers of Computer Science*.

Jiang, P.-T.; Zhang, C.-B.; Hou, Q.; Cheng, M.-M.; and Wei, Y. 2021. LayerCAM: Exploring Hierarchical Class Activation Maps for Localization. *IEEE Transactions on Image Processing*, 5875–5888.

Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; and Liu, Q. 2019. TinyBERT: Distilling BERT for Natural Language Understanding. In *Findings*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature*, 521(7553): 436–444.

Lenc, K.; and Vedaldi, A. 2014. Understanding Image Representations by Measuring Their Equivariance and Equivalence. *International Journal of Computer Vision*, 127: 456 – 476.

LI, C.; MAO, Y.; ZHANG, R.; and HUAI, J. 2020. A revisit to MacKay algorithm and its application to deep network compression. *Frontiers of Computer Science*, 14(4): 144304.

LI, S.-Y.; ZHENG, Y.-X.; SHI, Y.; HUANG, S.-J.; and CHEN, S. 2023. KD-Crowd: A Knowledge Distillation Framework for Learning from Crowds. *Frontiers of Computer Science*.

Li, Y.; Chen, Y.; Dai, X.; Chen, D.; Liu, M.; Yuan, L.; Liu, Z.; Zhang, L.; and Vasconcelos, N. 2020. MicroNet: Towards Image Recognition with Extremely Low FLOPs. *ArXiv*, abs/2011.12289.

Li, Y.; Hu, J.; Wen, Y.; Evangelidis, G.; Salahi, K.; Wang, Y.; Tulyakov, S.; and Ren, J. 2022a. Rethinking Vision Transformers for MobileNet Size and Speed. *ArXiv*, abs/2212.08059.

Li, Y.; Yuan, G.; Wen, Y.; Hu, E.; Evangelidis, G.; Tulyakov, S.; Wang, Y.; and Ren, J. 2022b. EfficientFormer: Vision Transformers at MobileNet Speed. *ArXiv*, abs/2206.01191.

Mehta, S.; and Rastegari, M. 2021. MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. *ArXiv*, abs/2110.02178.

Pan, Z.; Cai, J.; and Zhuang, B. 2023. Stitchable Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 16102–16112.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015a. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 211–252.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015b. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115: 211–252.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2020. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, 336–359.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; and Jégou, H. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, 10347–10357. PMLR.

Vanschoren, J. 2018. Meta-Learning: A Survey. *arXiv: Learning,arXiv: Learning*.

Wang, Q.; Yang, X.; Lin, S.; and Geng, X. 2023. Learngene: Inheriting Condensed Knowledge from the Ancestry Model to Descendant Models. *ArXiv*, abs/2305.02279.

Wang, Q.-F.; Geng, X.; Lin, S.-X.; Xia, S.-Y.; Qi, L.; and Xu, N. 2022. Learngene: From open-world to your learning task. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 8557–8565.

Wightman, R. 2019. PyTorch Image Models. https://github.com/rwightman/pytorch-image-models.

Yang, X.; Daquan, Z.; Liu, S.; Ye, J.; and Wang, X. 2022. Deep Model Reassembly. *ArXiv*, abs/2210.17409.

Zeiler, M. D.; and Fergus, R. 2013. Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision*.

Zhang, C.; Bengio, S.; and Singer, Y. 2019. Are All Layers Created Equal? *ArXiv*, abs/1902.01996.

Zhao, B.; Cui, Q.; Song, R.; Qiu, Y.; and Liang, J. 2022. Decoupled Knowledge Distillation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11943–11952.