Transformer as Linear Expansion of Learngene

Shiyu Xia, Miaosen Zhang, Xu Yang*, Ruiming Chen, Haokun Chen, 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
{shiyu_xia, 230228501, 101013120, 213193308, chenhaokun, xgeng}@seu.edu.cn

Abstract

We propose expanding the shared Transformer module to produce and initialize Transformers of varying depths, enabling adaptation to diverse resource constraints. Drawing an analogy to genetic expansibility, we term such module as *learngene*. To identify the expansion mechanism, we delve into the relationship between the layer's position and its corresponding weight value, and find that *linear function* appropriately approximates this relationship. Building on this insight, we present **T**ransformer as **L**inear **E**xpansion of learn**G**ene (**TLEG**), a novel approach for flexibly producing and initializing Transformers of diverse depths. Specifically, to learn learngene, we firstly construct an auxiliary Transformer linearly expanded from learngene, after which we train it through employing soft distillation. Subsequently, we can produce and initialize Transformers of varying depths via linearly expanding the well-trained learngene, thereby supporting diverse downstream scenarios. Extensive experiments on ImageNet-1K demonstrate that TLEG achieves comparable or better performance in contrast to many individual models trained from scratch, while reducing around 2× training cost. When transferring to several downstream classification datasets, TLEG surpasses existing initialization methods by a large margin (*e.g.*, +6.87% on iNat 2019 and +7.66% on CIFAR-100). Under the situation where we need to produce models of varying depths adapting for different resource constraints, TLEG achieves comparable results while reducing around 19× parameters stored to initialize these models and around 5× pre-training costs, in contrast to the pre-training and fine-tuning approach. When transferring a fixed set of parameters to initialize different models, TLEG presents better flexibility and competitive performance while reducing around 2.9× parameters stored to initialize, compared to the pre-training approach.

Introduction

Deep neural networks (DNNs), *e.g.*, Vision Transformer, have demonstrated remarkable performance in a wide variety of computer vision tasks (Sun et al. 2019; Carion et al. 2020; Liang et al. 2020; Dosovitskiy et al. 2021; Zhang et al. 2021; Qin, Zhang, and Tang 2023). Parameter initialization is a pivotal step prior to training and wields a critical influence over the ultimate quality of the trained network (Glorot and Bengio 2010; He et al. 2016; Arpit, Campos, and Bengio 2019; Huang et al. 2020; Zhang, Bao, and Ma 2021; Czyzewski, Nowak, and Piechowiak 2022). Nowadays, large-scale pre-training on massive curated data brings huge *foundation models*, which furnishes a superb starting point for fine-tuning across diverse downstream tasks (Liu et al. 2021; Oquab et al. 2023). However, the parameters of original whole model are required storing and updating separately for each downstream task during the popular pre-training and fine-tuning process, which is prohibitively expensive and time-consuming for the current ever-increasing capacity of vision models. Furthermore, this approach lacks the flexibility to initialize models of *varying scales* to meet diverse scenario demands, such as edge and IoT devices with constrained computational resources. Therefore, in different application scenarios, a fundamental research question naturally arises: *how to efficiently produce and initialize individual models considering both the model performance and resource constraint?*

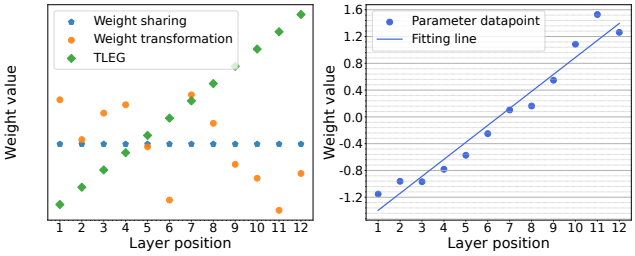


Figure 1: Left: the relationship between the layer's position and its corresponding weight value of different methods. Right: our empirical observation of such relationship based on ViT-B, which shows approximate linear relationship.

Mimicking the behaviour of the organismal gene, (Wang et al. 2022a, 2023) proposed an innovative learning framework known as *Learngene* which firstly learns the condensed knowledge termed as **learngene** from the ancestry model, and then inherits this small part to initialize descendant/-downstream models. The existing work He-LG (Wang et al. 2022a) extracts a few integral layers as learngene based on the gradient information of the ancestry model, after which the descendant models are constructed by stacking the randomly initialized low-level layers with the extracted learngene layers. Nevertheless, there are three major limitations existed in (Wang et al. 2022a). Firstly, the strategies of ex-

*Co-corresponding author.

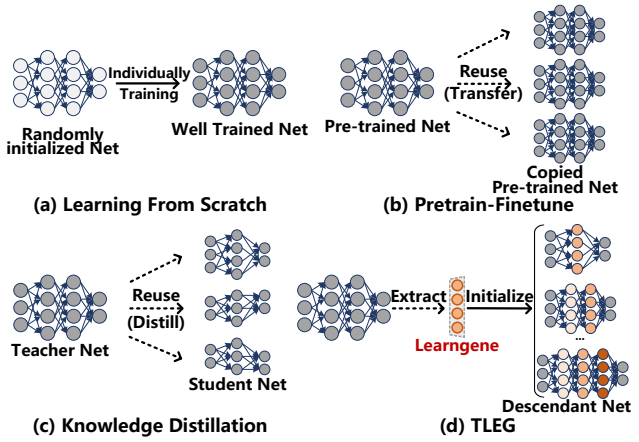


Figure 2: (a) Learning from scratch randomly initializes different networks in varied applications, where the training and storage costs increase linearly with the number of possible cases. (b) Pretrain-Finetune stores and reuses the original whole model every time facing different scenarios. (c) Distillation transfers knowledge from a large teacher net to a smaller student one, which requires forward propagation on teacher every time training new students. (d) Our TLEG directly extracts one compact learngene from an ancestry net *once* and quickly initializes new descendant nets with our linear expansion strategy, allowing adaptation to diverse resource constraints.

tracting and utilizing the learngene are inconsistent, yielding diminished performance. Secondly, He-LG ignores descendant models across different scales. Lastly, He-LG does not explore Transformer-based architectures, with which the performance is also unsatisfactory.

As mentioned before, Learngene is dedicated to retaining the most generalizable part of the ancestry model, which naturally directs our attention towards eliminating redundant parameters. One prominent approach that exemplifies this endeavor is weight sharing (Lan et al. 2020; Zhang et al. 2022), which shares identical parameters across all layers to maximise parameter elimination. Despite its simplicity, such fully-sharing method notably compromises the model capabilities (Zhang et al. 2022). To alleviate this problem, researchers (Zhang et al. 2022) apply weight transformation, which imposes learnable functions on the shared weights to increase parameter diversity. Interestingly, if we treat the parameters of each layer as one high-dimensional tensor, we can illustrate the relationship between the layer's position and its corresponding parameter value, as shown in the left portion of Fig. 1. Specifically, weight sharing presents a "horizontal line" as each layer shares identical parameters. Correspondingly, weight transformation (Zhang et al. 2022) scatters the parameters due to the layer-specific mapping function. Upon closer observation, we wonder if there exists a intermediate situation between them, *i.e.*, is there any simpler function that could approximate the relationship between the layer's position and its corresponding parameter value?

To obtain some empirical observations of such relationship, we use PCA (Karamizadeh et al. 2013) to transform each tensor to 1-D data point for convenience. Here we choose the well-trained ViT-B (Dosovitskiy et al. 2021) for analyzing. Please see more details and visualizations in the appendix. As shown in the right portion of Fig. 1, a noteworthy observation emerges: most data points do not exhibit irregular arrangements, instead they manifest *an approximately linear trend*. Among the multitude of fitting functions, the linear function stands out as the simplest yet effective one for approximating this trend. Inspired by this insight, we present **T**ransformer as **L**inear **E**xpansion of learn**G**ene (**TLEG**), a novel approach for elastic Transformer production and initialization. Specifically, we adopt linear expansion on two shared parameter modules, *i.e.*, $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$, both of which compose learngene $\theta_{\mathcal{LG}}$, to produce the parameters of each Transformer layer θ_l:

$$\theta_l = \theta_{\mathcal{B}} + \frac{l-1}{L} \times \theta_{\mathcal{A}}, \quad l = 1, 2, ..., L, \tag{1}$$

where L denotes the total number of layers.

To learn the learngene parameters $\theta_{\mathcal{LG}}$, we design an auxiliary Transformer network (Aux-Net) where each layer is linearly expanded from $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$ based on Eq. (1). To ensure clarity, we exemplify the construction process using a 4-layer Aux-Net as an example. The parameters of the first layer are formulated as $\theta_1 = \theta_{\mathcal{B}} + \frac{1-1}{4} \times \theta_{\mathcal{A}}$. Correspondingly, the parameters of the second layer are formulated as $\theta_2 = \theta_{\mathcal{B}} + \frac{2-1}{4} \times \theta_{\mathcal{A}}$, and so forth for subsequent layers. Then we proceed to train the Aux-Net by employing distillation technique (Hinton, Vinyals, and Dean 2015), which enables knowledge condensation from a large ancestry model. Despite the Aux-Net containing four layers, the linear constraint always holds during training, which means that only $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$ will be updated throughout the training process.

After obtaining learngene containing well-trained $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$, we can produce and initialize descendant models (Des-Net) of varying depths adapting for different resource constraints. For example, a shallow network can be deployed in the lightweight edge device and a deeper one can be supported in a computation center equipped with ample computation resources. To enhance clarity, we provide an example of initializing a 6-layer Des-Net. In this instance, the parameters of the first layer can be initialized as $\theta_1 = \theta_{\mathcal{B}} + \frac{1-1}{6} \times \theta_{\mathcal{A}}$, similarly the parameters of the second layer would be $\theta_2 = \theta_{\mathcal{B}} + \frac{2-1}{6} \times \theta_{\mathcal{A}}$, and so forth. Notably, we only employ linear expansion strategy to initialize these Des-Nets, after which they undergo standard fine-tuning procedure. Our main **contributions** are summarized as follows:

- We empirically discover an approximately linear relationship between the position of a layer and its corresponding weight value within well-trained Transformer models.
- Taking inspiration from above observations, we propose a coherent approach termed TLEG for efficient model construction, which linearly expands learngene to produce and initialize Transformers across a spectrum of scales.
- Extensive experiments demonstrate the effectiveness and efficiency of TLEG, *e.g.*, compared to training different models from scratch, training with a compact learngene can obtain on-par or better performance while reducing large training costs.

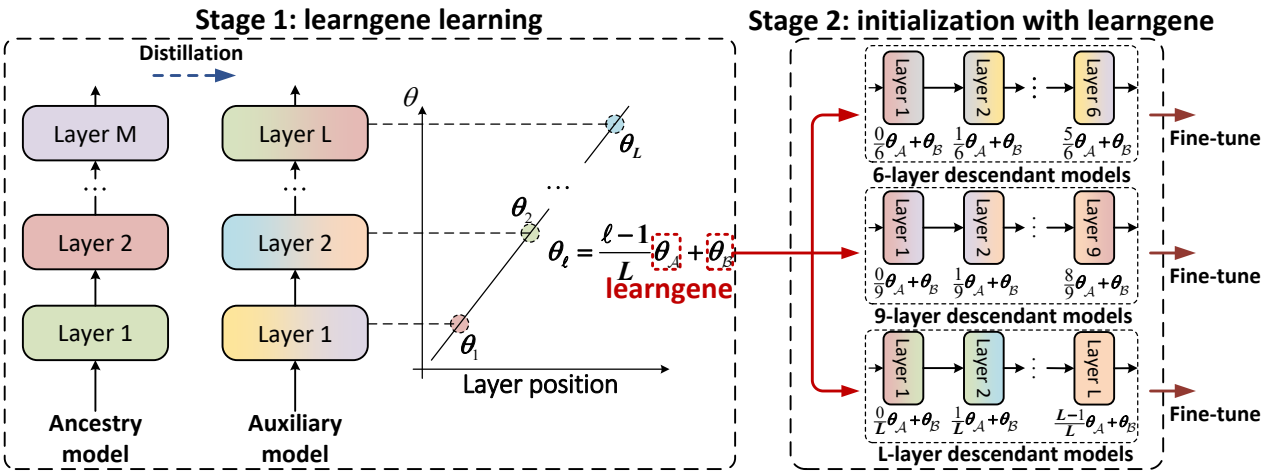


Figure 3: In the first stage, we construct an auxiliary model wherein each layer is linearly expanded from learngene. Subsequently, we train it through distillation. After obtaining learngene with well-trained $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$, in the second stage, we initialize descendant models of varying depths via adopting linear expansion on $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$, enabling adaptation to diverse resource constraints. Lastly, the descendant models are fine-tuned normally without the restriction of linear expansion.

Related Work

Parameter Initialization

Parameter initialization constitutes an important step prior to model training and plays a crucial role in boosting the model quality (Arpit, Campos, and Bengio 2019; Huang et al. 2020; Czyzewski, Nowak, and Piechowiak 2022). Proper initialization has been proved to improve the efficiency of model training (LeCun et al. 2002), whereas arbitrary initialization may impede the optimization process (Mishkin and Matas 2015). Extensive initialization approaches have been proposed for models trained from scratch, such as random initialization, xavier initialization (Glorot and Bengio 2010), kaiming initialization (He et al. 2016) and self-distillation (Zhang, Bao, and Ma 2021). Nowadays, large-scale pre-training on massive curated data provides an excellent initialization for fine-tuning models across a spectrum of downstream tasks (Jia et al. 2021; Radford et al. 2021; Oquab et al. 2023). However, such scheme needs to *reuse the original whole model* every time facing different downstream tasks regardless of the resources available to those tasks, as shown in Fig. 2(b). More importantly, we need to pre-train again when meets another model with different scales, which is extremely time consuming and computationally expensive. In contrast, we propose training learngene *once* which can be linearly expanded to cover a fine-grained level of model complexity/performance for a wide range of deployment scenarios.

Knowledge Distillation

There exists extensive literature studying knowledge distillation (Jiao et al. 2020; Wang et al. 2020; Gou et al. 2021; Wu et al. 2022; Ren et al. 2023; Ji et al. 2023; Li et al. 2023). DeiT (Touvron et al. 2021) introduces a distillation token to allow the vision transformer to learn from a ConvNet teacher. MiniViT (Zhang et al. 2022) applies weight distillation to transfer knowledge from large-scale models to weight-multiplexed models. TinyMIM (Ren et al. 2023) studies the distillation framework for masked image modeling pretrained vision transformers. What they have in common is that distillation requires additional forward passes through a pretrained teacher every time training a new student, which inevitably consumes extra resources for storage and computation of teacher models, as shown in Fig. 2(c). In contrast, we distill rich knowledge from the pretrained ancestry model to learngene *once*, after which we can produce models of diverse scales while getting rid of the ancestry model.

Weight Sharing

Weight sharing is a simple but effective strategy to solve over-parameterization problem (Bai, Kolter, and Koltun 2019; Kovaleva et al. 2019) in large pretrained Transformers (Devlin et al. 2018). By contrast, our proposed linear expansion strategy promotes parameter diversity of each layer while preserving parameter efficiency.

Approach

Fig. 3 depicts the pipeline of TLEG. In stage 1, an auxiliary model is constructed to help learn learngene parameters $\theta_{\mathcal{LG}} = \{\theta_{\mathcal{A}}, \theta_{\mathcal{B}}\}$, where each layer is linearly expanded from $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$. The auxiliary model is trained by distilling knowledge from the ancestry model and note that during training, the linear constraint always holds in the auxiliary model, *i.e.*, only $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$ are trained. In stage 2, the well-trained $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$ are linearly expanded to initialize descendant models of varying depths. Lastly, the descendant models are fine-tuned normally without the restriction of linear expansion. Next, we briefly introduce some preliminaries.

Preliminaries

Witnessing the remarkable performance of vision transformer (ViT) (Dosovitskiy et al. 2021) and its variants in diverse

vision tasks (Bao et al. 2022; Wang et al. 2022b), we explore learngene based on ViT. ViT firstly splits an image into a few patches and maps them into D-dimensional patch embeddings. Then position embeddings are added to them to get N input embeddings $Z_0 \in \mathbb{R}^{N\times D}$. A ViT encoder stacks a few layers each containing multi-head self-attention (MSA) and multi-layer perceptron (MLP). Let h denote the number of heads where each head performs self-attention. In the k^{th} head, we linearly generate the queries Q_k, keys K_k and values $V_k \in \mathbb{R}^{N\times d}$ with parameter matrices W_k^Q, W_k^K and $W_k^V \in \mathbb{R}^{D\times d}$, where d is the projected dimension of each head. We denote the attention output of head k as $A_k(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d}})V$. MSA jointly deals with information from different embedding subspaces as $MSA(Q,K,V) = Concat(head_1, ..., head_h)W^O$, where $head_k = A_k(Q_k, K_k, V_k)$, $W^O \in \mathbb{R}^{hd\times D}$ and $Concat(\cdot)$ means the catenation of the outputs of all heads.

Besides, each layer contains a MLP block which consists of two linear transformations with a GELU (Hendrycks and Gimpel 2016) activation. We denote the MLP output as $MLP(x) = \sigma(xW^1 + b^1)W^2 + b^2$, where $W^1 \in \mathbb{R}^{D\times D_h}$, $b^1 \in \mathbb{R}^{D_h}$, $W^2 \in \mathbb{R}^{D_h\times D}$ and $b^2 \in \mathbb{R}^D$ represent the weights and biases for the two linear transformations, respectively. $\sigma(\cdot)$ denotes the activation function. Usually we set $D_h > D$. Layer normalization (LN) (Ba, Kiros, and Hinton 2016) and residual connections are employed before and after every block. We denote the LN output as $LN(x) = \frac{x-\mu}{\delta} \circ \gamma + \beta$, where μ and δ are the mean and standard deviation of the embeddings respectively, $\circ$ means the element-wise dot, $\gamma \in \mathbb{R}^D$ and $\beta \in \mathbb{R}^D$ are learnable transform parameters.

Linear Expansion of Learngene

As mentioned before, we adopt linear expansion on the two shared parameter modules, namely $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$, each of which contains the parameters of an entire Transformer layer. Since the learngene $\theta_{\mathcal{LG}}$ comprises $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$, it inherently encompasses the parameters of two complete Transformer layers. Take the 12-layer ViT-B (87M) (Dosovitskiy et al. 2021) as an example, $\theta_{\mathcal{LG}}$ comprises approximately 14.7M parameters which is equivalent to the number of parameters of two layers. Next, we elaborate on the linear expansion of each component, *i.e.*, MSA, MLP and LN, within one layer.

Linear Expansion of MSA. Based on our empirical observations, we linearly expand the learnable parameter matrices in MSA module. Formally, we linearly expand parameter matrices W_k^Q, W_k^K, W_k^V and W^O through Eq. (1). Take W_k^Q as an example, its linearly expanded version is:

$$W_k^Q = W_{k(\mathcal{B})}^Q + \frac{l-1}{L} \times W_{k(\mathcal{A})}^Q, \quad l = 1, 2, ..., L, \quad (2)$$

where $W_{k(\mathcal{A})}^Q$ and $W_{k(\mathcal{B})}^Q$ are the corresponding learngene parameters of MSA in $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$ respectively, L denotes the total number of layers. Such linear expansion can make parameters linearly different across layers while preserving the common knowledge during the training process.

Linear Expansion of MLP. We further impose linear expansion on MLP to preserve the common knowledge while improving parameter diversity. In particular, we linearly expand parameter matrices W^1, b^1, W^2 and b^2 to obtain the linearly expanded version through Eq. (1), *e.g.*, the linearly expanded version of W^1 is:

$$W^1 = W_{(\mathcal{B})}^1 + \frac{l-1}{L} \times W_{(\mathcal{A})}^1, \quad l = 1, 2, ..., L, \quad (3)$$

where $W_{(\mathcal{A})}^1$ and $W_{(\mathcal{B})}^1$ are the corresponding learngene parameters of MLP in $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$.

Linear Expansion of LN. Lastly, we linearly expand learnable parameters γ and β through Eq. (1). Take γ as an example, its linearly expanded version is:

$$\gamma = \gamma_{(\mathcal{B})} + \frac{l-1}{L} \times \gamma_{(\mathcal{A})}, \quad l = 1, 2, ..., L, \quad (4)$$

where $\gamma_{(\mathcal{A})}$ and $\gamma_{(\mathcal{B})}$ are the corresponding learngene parameters of LN in $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$.

Learning Strategy of Learngene

The learngene $\theta_{\mathcal{LG}}$ is used to construct the MSA, MLP and LN blocks by Eq. (2) to Eq. (4), while an integral Transformer model also requires some other components like the patch projection and task-specific head. Thus we also add them to build the auxiliary model (**Aux-Net**), after which we train it through employing distillation. For simplicity, we only consider penalizing output discrepancy (Hinton, Vinyals, and Dean 2015) between the ancestry model and auxiliary model. Additional distillation techniques (Zhang et al. 2022; Ren et al. 2023) can also be seamlessly integrated into our training process, thereby further boosting the quality of the trained learngene. Noteworthy, the linear constraint in Eq. (2) to Eq. (4) always exists during training. For example, the update of W_k^Q in Eq. (2) of each layer finally leads to the update of $W_{k(\mathcal{B})}^Q$ and $W_{k(\mathcal{A})}^Q$. Therefore, although Aux-Net contains L layers, only $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$ are trained during distillation.

Soft Distillation. (Hinton, Vinyals, and Dean 2015) proposes to minimize the KL-divergence between the probability distributions over their output predictions of the teacher model and that of the student one. We leverage such strategy to introduce one distillation loss:

$$\mathcal{L}_D = KL(\phi(z_s/\tau), \phi(z_t/\tau)), \quad (5)$$

where z_t means the logits output of the pretrained ancestry model (*e.g.*, Levit-384 (Graham et al. 2021)), z_s means the logits output of the auxiliary model, τ means the temperature for distillation, ϕ means the softmax function and $KL(\cdot,\cdot)$ means KL-divergence loss function. Combined with the classification loss, our total training loss is defined as:

$$\mathcal{L} = (1-\lambda)CE(\phi(z_s), y) + \lambda\mathcal{L}_D, \quad (6)$$

where y means ground-truth label, $CE(\cdot,\cdot)$ means cross-entropy loss function and λ means the trade-off coefficient.

Initialization with Learngene

After obtaining learngene consisting of well-trained $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{B}}$, we can produce multiple descendant models (**Des-Net**) of varying depths, catering to diverse deployment scenarios.

Model	D	L^{ds}	Params (M)	FLOPs (G)	Scratch Top-1(%)	TLEG Top-1(%)
Des-Ti	192	3	1.7	0.3	45.0	46.6 **(+1.6)**
		6	3.1	0.6	56.9	58.2 **(+1.3)**
		9	4.4	0.9	62.3	62.5 **(+0.2)**
		12	5.7	1.3	65.2	65.4 **(+0.2)**
Des-S	384	3	6.1	1.2	56.2	57.1 **(+0.9)**
		4	7.9	1.6	62.0	63.7 **(+1.7)**
		5	9.6	1.9	67.3	67.5 **(+0.2)**
		6	11.4	2.3	68.7	69.5 **(+0.8)**
		7	13.2	2.7	70.6	71.1 **(+0.5)**
		8	15.0	3.1	71.7	72.3 **(+0.6)**
		9	16.7	3.5	73.0	73.2 **(+0.2)**
		10	18.5	3.8	73.8	73.9 **(+0.1)**
		11	20.3	4.2	75.5	75.4 (-0.1)
		12	22.1	4.6	75.0	75.1 **(+0.1)**
Des-B	768	3	22.8	4.5	65.3	66.3 **(+1.0)**
		4	29.9	5.9	70.4	71.6 **(+1.2)**
		5	37.0	7.4	73.5	74.4 **(+0.9)**
		6	44.0	8.8	75.4	76.2 **(+0.8)**
		7	51.1	10.3	76.5	77.3 **(+0.8)**
		8	58.2	11.7	77.2	78.1 **(+0.9)**
		9	65.3	13.1	78.0	78.7 **(+0.7)**
		10	72.4	14.6	78.2	79.1 **(+0.9)**
		11	79.5	16.0	79.0	79.6 **(+0.6)**
		12	86.6	17.5	78.6	79.9 **(+1.3)**

Table 1: Performance comparisons on ImageNet-1K between models trained from scratch and those initialized via TLEG.

Benefiting from the flexibility of our proposed linear expansion strategy, we can initialize descendant models of different L^{ds} by Eq. (1). Notably, different from the Aux-Net trained under the linear constraint, the descendant models are only initialized using Eq. (2) to Eq. (4). After initialization, this constraint is removed and all the parameters of the descendant models will be updated. For example, W_k^Q in Eq. (2) of different layers will be updated normally according to their corresponding gradients irrespective of the linear constraints.

Experiments

Experimental Setup

We conduct experiments on ImageNet-1K (Deng et al. 2009) and several middle/small-scale datasets including iNaturalist 2019 (**iNat 19**) (Zhou et al. 2020), Mini-ImageNet (**Mi-INet**) (Vinyals et al. 2016), Tiny-ImageNet (**Ti-INet**) (Le and Yang 2015), CIFAR-10 (**C-10**), CIFAR-100 (**C-100**) (Krizhevsky, Hinton et al. 2009) and Food-101 (**F-101**) (Bossard, Guillaumin, and Van Gool 2014). Model performance is measured by Top-1/5 accuracy (Top-1/5(%)). Furthermore, we report the FLOPs(G), Params(M) and S-Params(M) as indicators of theoretical complexity, the number of individual model parameters and parameters transferred/stored to initialize, respectively. We denote Aux-Ti/S/B as the variants of Aux-Net, in which we adopt linear expansion on MSA, MLP and LN compared to DeiT-Ti/S/B (Touvron et al. 2021). For Des-Net, we introduce Des-Ti/S/B where we change the number of layers based on DeiT-Ti/S/B. We firstly train Aux-Ti/S/B on ImageNet-1K to obtain learngenes, during which we choose Levit-384 (Graham et al. 2021) as the ancestry model to employ distillation. Then we initialize Des-Ti/S/B with learngenes and fine-tune them. Source code is available at https://github.com/AlphaXia/TLEG.

Main Results

TLEG achieves on-par or better performance with much less training efforts compared to training from scratch on ImageNet-1K. To validate the robustness of this claim, we conduct extensive experiments where different model settings, *e.g.*, different embedding dimensions and model depths are adopted. The ImageNet-1K classification performance of 24 different Des-Nets are reported in Table 1, where "TLEG" denotes the models initialized with our learngenes and "Scratch" denotes the randomly initialized models trained from scratch. As shown in Table 1, TLEG can cover a fine-grained level of model complexity, while achieving comparable or better performance and significantly improving training efficiency. For Aux-S and Des-S of 10 different depths, we train Aux-S for 150 epochs and each Des-S for 35 epochs, except that we train 11-layer Des-S for 45 epochs. From Table 1, we observe that TLEG achieves competitive performance and reduces around **2**$\times$ training costs (10$\times$100 epochs *vs*. 150+9$\times$35 epochs+1$\times$45 epochs), in contrast to training each Des-S from scratch for 100 epochs. For Aux-B and Des-B of 10 different depths, we train Aux-B for 100 epochs and each Des-B for 40 epochs. From Table 1, we find that TLEG achieves better performance and reduces around **2**$\times$ training costs (10$\times$100 epochs *vs*. 100+10$\times$40 epochs), compared to training each Des-B from scratch for 100 epochs. For Aux-Ti and Des-Ti of 4 different depths, we train Aux-Ti for 150 epochs and each Des-Ti for 50 epochs. From Table 1, we observe that TLEG achieves better performance and reduces a few training costs (4$\times$100 epochs *vs*. 150+4$\times$50 epochs), contrary to training each Des-Ti from scratch for 100 epochs. Overall, the efficiency of TLEG becomes more evident with the number of Des-Nets increasing as we only need to train our learngenes *once*.

TLEG provides competitive results when transferring to a wide range of downstream classification datasets. We compare TLEG against training from scratch and pre-training method whose performance is regarded as upper-bound on 6 classification datasets. Moreover, we adopt state-of-the-art compression methods to our setting. Specifically, (1) Scratch. We train Des-Nets from scratch on the downstream datasets. (2) Pre-Fin(U). We pretrain each Des-Net on ImageNet-1K with 100 epochs. (3) Mini-Init (Zhang et al. 2022). We pretrain Mini-DeiT on ImageNet-1K with 100 epochs, where the number of shared part is 6. Then we use the shared parts to initialize Des-Nets. (4) Share-Init (Lan et al. 2020). We pre-train DeiT (Touvron et al. 2021) on ImageNet-1K with 100 epochs, where we share the parameters of each layer. Then we use the shared part to initialize Des-Nets. (5) He-LG (Wang et al. 2022a). We extract last 3 layers of pretrained

Model	Params(M)	Method	S-Params(M)	iNat 19	Mi-INet	Ti-INet	C-100	C-10	F-101
Des-Ti	3.0	Pre-Fin(U)	2.9	58.12	77.00	66.32	80.81	96.65	83.24
		Scratch	0	37.16	60.37	58.24	67.44	88.30	61.54
		He-LG	1.4	41.55	65.74	63.11	70.19	91.66	72.54
		Share-Init	0.6	42.58	63.88	60.98	71.23	92.56	67.44
		Mini-Init	2.8	51.22	70.26	61.51	74.01	93.07	77.36
		TLEG(ours)	**1.0**	**55.64**	**74.07**	**65.02**	**78.66**	**95.32**	**82.80**
Des-S	11.3	Pre-Fin(U)	11.0	68.48	81.78	72.24	84.43	97.59	87.80
		Scratch	0	50.79	55.73	61.24	73.32	92.49	74.64
		He-LG	5.3	53.21	59.87	62.37	78.13	93.12	77.09
		Share-Init	2.1	54.14	61.64	63.12	74.08	94.15	78.11
		Mini-Init	11.0	59.83	73.39	64.56	75.98	93.67	81.79
		TLEG(ours)	**3.9**	**66.70**	**80.92**	**71.32**	**83.64**	**97.68**	**87.27**

Table 2: Performance comparisons on middle/small-scale datasets when transferring pretrained parameters (S-Params(M)) to initialize 6 layer Des-Ti/S. Here, Params(M) means the average number of individual model parameters on different datasets.

Model	L^{ds}	Params (M)	Pre-Fin(U) S-P(M)	Pre-Fin(U) Top-1(%)	TLEG S-P(M)	TLEG Top-1(%)
Des-B	4	29.2	28.8	87.01	14.7	86.52
	6	43.3	43.0	87.45		87.03
	8	57.6	57.1	88.03		87.96
	10	71.7	71.3	88.12		88.21
	12	85.9	85.5	88.62		88.34

Table 3: Comparisons on C-100 of Des-B with different layer numbers. For Pre-Fin(U), S-P(M) means the number of pretrained parameters used to initialize, which totally requires 285.7M. However, TLEG only preserves 14.7M parameters to initialize all listed Des-B, which reduces the number of parameters stored for initialization by 19× (285.7M *vs*. 14.7M).

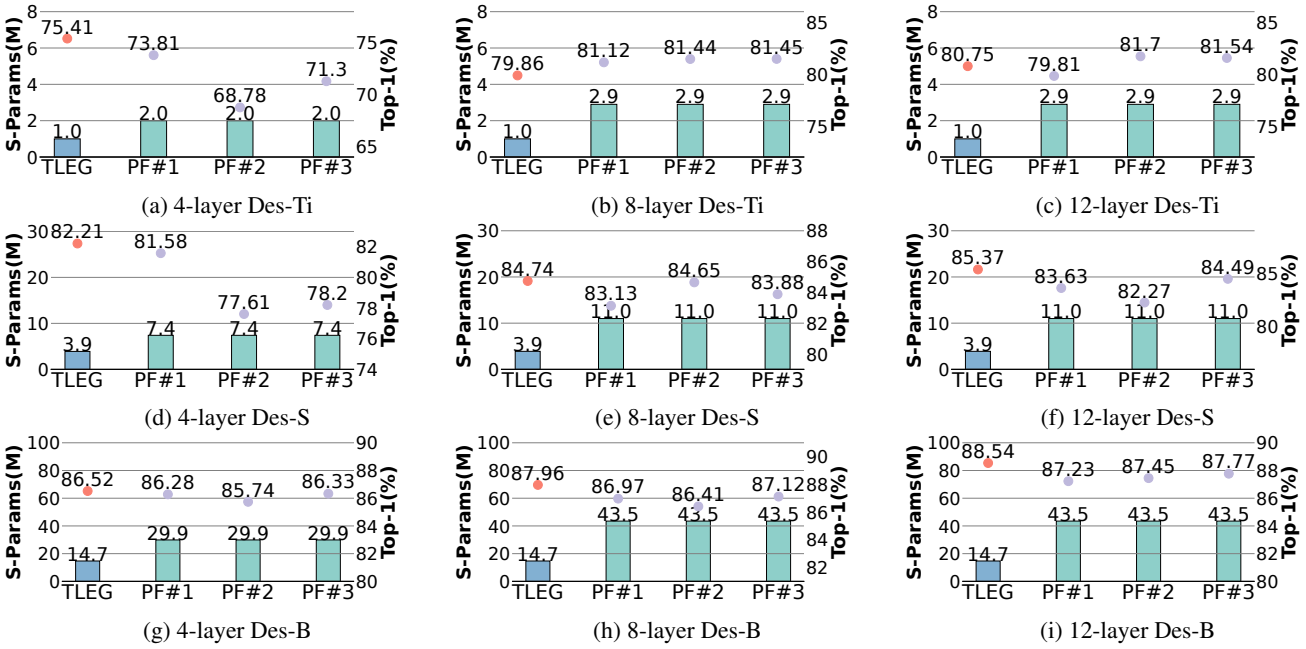
Method	#	MSA	MLP	LN	Top-1 (%)	Top-5 (%)
Pre-Fin(U)	1				84.43	96.39
TLEG	2	✓			79.16	95.17
	3		✓		77.26	94.29
	4	✓	✓		82.03	95.74
	5	✓	✓	✓	**83.64**	**96.53**

Table 4: Performance of 6-layer Des-S on C-100 when we employ linear expansion strategy on different modules in the 6-layer Aux-S. #1 means the pre-training and fine-tuning scheme, which transfers the parameters of the total model to initialize Des-S. #2/#3/#4/#5 means linearly expand MSA / MLP / MSA and MLP / MSA, MLP and LN respectively.

DeiTs and stack them with randomly-initialized low-level layers to produce Des-Nets. For (2)-(5), we finetune Des-Nets on the downstream datasets. As shown in Table 2, TLEG achieves performance gains compared with several baselines, which verifies the effectiveness of initialization with learngenes. For example, we observe that TLEG outperforms Mini-Init by $\mathbf{7.53\%}$, $\mathbf{6.76\%}$, and $\mathbf{7.66\%}$ respectively on Mi-INet, Ti-INet and C-100 with Des-S, whereas TLEG reduces **2.8**× parameters used to initialize (11.0M *vs*. 3.9M). Moreover, TLEG achieves comparable performance compared with upper-bound method Pre-Fin(U), showing that the common knowledge, *i.e*., learngene, is satisfactorily learned and used to initialize Des-Nets.

TLEG significantly reduces the parameters stored to initialize and pre-training costs compared with Pre-Fin(U) when initializing diverse models. We compare 5 different Des-B initialized from learngenes to those initialized via Pre-Fin(U), where the performance of latter is regarded as upper-bound. In Table 3, TLEG achieves comparable performance and efficiently initializes diverse models with fewer storage costs. Specifically, TLEG substantially reduces **19**× (285.7M *vs*. 14.7M) parameters stored to initialize, compared to Pre-Fin(U). Moreover, Pre-Fin(U) needs to pretrain each different Des-B individually, while TLEG only requires training learngene *once*, thus substantially reducing the pre-training costs. Specifically, TLEG reduces **5**× (5×100 epochs *vs*. 1×100 epochs) pre-training costs compared to Pre-Fin(U) when facing 5 different Des-Nets. Notably, the efficiency of TLEG becomes more obvious because the pre-training costs of Pre-Fin(U) increase with the number of different Des-Nets.

TLEG presents better performance and flexibility when initializing models of different scales with a fixed set of parameters. Specifically, we have three 6-layer model of different embedding dimensions pretrained on Imagenet-1K via Pre-Fin(U) with 57.4M (2.9+11.0+43.5M) parameters and relatively smaller learngenes with 19.6M (1.0+3.9+14.7M) parameters. Now we need to initialize the 4-layer, 8-layer and 12-layer Des-Ti/S/B. For TLEG, we initialize them via linearly expanding the learned learngenes conveniently. For Pre-Fin(U), we have several intuitive choices: For the 8-layer and 12-layer Des-Ti/S/B, Pre-Fin #1/#2/#3 means we initialize the first/last/middle 6 layer of them with the pretrained 6-layer models. For the 4-layer Des-Ti/S/B, Pre-Fin #1/#2/#3 means that we use the first/last/middle 4 layer of 6-layer pretrained models to initialize them. As shown in Fig. 4, we observe that TLEG achieves comparable or even superior



(a) 4-layer Des-Ti (b) 8-layer Des-Ti (c) 12-layer Des-Ti

(d) 4-layer Des-S (e) 8-layer Des-S (f) 12-layer Des-S

(g) 4-layer Des-B (h) 8-layer Des-B (i) 12-layer Des-B

Figure 4: Performance on C-100 when initializing diverse Des-Nets of different scales with a fixed set of parameters.

Model	Part	Top-1 (%)	Top-5 (%)
Des-Ti	1 - 3	74.29	93.58
	4 - 6	72.00	92.55
	1 - 6	**78.66**	**95.35**
Des-S	1 - 3	74.88	93.59
	4 - 6	74.33	93.33
	1 - 6	**83.64**	**96.53**

Table 5: Performance of 6-layer Des-Ti/S on C-100 under partially initialization.

Model	Type	Top-1 (%)	Top-5 (%)
Des-Ti	All	78.66	95.35
	Partial	**79.02**	**96.12**
Des-S	All	83.64	96.53
	Partial	**84.13**	**96.72**

Table 6: Performance of 6-layer Des-Ti/S on C-100 with different linear expansion strategies.

performance over Pre-Fin(U) while reducing around **2.9**× (57.4M *vs*. 19.6M) parameters stored to initialize. For example, TLEG outperforms the best variant of Pre-Fin(U) by $\mathbf{0.63\%}$, $\mathbf{0.09\%}$ and $\mathbf{0.88\%}$ respectively on 4-layer, 8-layer and 12-layer Des-S. Overall, when initializing different models with a fixed set of parameters, TLEG demonstrates better flexibility and performance, showing that learngene contains generalizable knowledge and serves as a great starting point for training Des-Nets of diverse scales.

Ablation and Analysis

We investigate the performance of Des-Nets when we (1) adopt linear expansion on different modules in Aux-Nets, (2) initialize partial layers of Des-Nets, (3) adopt linear expansion on partial layers in Aux-Nets.

The effect of different linearly expanded modules. We apply linear expansion strategy on different modules of Aux-S to achieve several variants. Then we utilize the linearly expanded module to initialize corresponding module in Des-S and randomly initialize other modules. As shown in Table 4, we can observe that #5 achieves the best accuracy, which is comparable against Pre-Fin(U).

The effect of initializing partial layers of Des-Nets. We initialize partial layers of 6-layer Des-S. As shown in Table 5, we choose to initialize first half (1-3), second half (4-6) and all (1-6) of the layers. We observe that initializing all layers achieves the best performance.

The effect of partial linear expansion. We adopt linear expansion on partial layers, *i.e*., from the 3rd layer to the last layer, in 6-layer Aux-Ti/S to learn learngene. As shown in Table 6, we observe that adopting partial linear expansion achieves slightly better performance. Nevertheless, we adopt linear expansion on all layers in our main experiments. More variants of linear expansion strategies are left for future work.

Conclusion

In this paper, we proposed a new approach termed TLEG to produce and initialize Transformers of varying depths via linearly expanding learngene, enabling adaptation to diverse real-world applications containing different resources. Experimental results under various model initialization settings demonstrated the effectiveness and flexibility of TLEG.

Acknowledgements

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

References

Arpit, D.; Campos, V.; and Bengio, Y. 2019. How to initialize your network? robust initialization for weightnorm & resnets. *Advances in Neural Information Processing Systems*, 32.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bai, S.; Kolter, J. Z.; and Koltun, V. 2019. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32.

Bao, H.; Dong, L.; Piao, S.; and Wei, F. 2022. Beit: Bert pre-training of image transformers. *ICLR*.

Bossard, L.; Guillaumin, M.; and Van Gool, L. 2014. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, 446–461. Springer.

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, 213–229. Springer.

Czyzewski, M. A.; Nowak, D.; and Piechowiak, K. 2022. Breaking the Architecture Barrier: A Method for Efficient Knowledge Transfer Across Networks. *arXiv preprint arXiv:2212.13970*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256. JMLR Workshop and Conference Proceedings.

Gou, J.; Yu, B.; Maybank, S. J.; and Tao, D. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129: 1789–1819.

Graham, B.; El-Nouby, A.; Touvron, H.; Stock, P.; Joulin, A.; Jégou, H.; and Douze, M. 2021. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, 12259–12269.

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.

Hendrycks, D.; and Gimpel, K. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Huang, X. S.; Perez, F.; Ba, J.; and Volkovs, M. 2020. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, 4475–4483. PMLR.

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*, 17(2): 172312.

Jia, C.; Yang, Y.; Xia, Y.; Chen, Y.-T.; Parekh, Z.; Pham, H.; Le, Q.; Sung, Y.-H.; Li, Z.; and Duerig, T. 2021. Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning*, 4904–4916. PMLR.

Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; and Liu, Q. 2020. Tinybert: Distilling bert for natural language understanding. *EMNLP*.

Karamizadeh, S.; Abdullah, S. M.; Manaf, A. A.; Zamani, M.; and Hooman, A. 2013. An overview of principal component analysis. *Journal of Signal and Information Processing*, 4(3B): 173.

Kovaleva, O.; Romanov, A.; Rogers, A.; and Rumshisky, A. 2019. Revealing the dark secrets of BERT. *EMNLP*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. *Technique Report*.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2020. Albert: A lite bert for self-supervised learning of language representations. *ICLR*.

Le, Y.; and Yang, X. 2015. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7): 3.

LeCun, Y.; Bottou, L.; Orr, G. B.; and Müller, K.-R. 2002. Efficient backprop. In *Neural networks: Tricks of the trade*, 9–50. Springer.

Li, S.; Zheng, Y.; Shi, Y.; Huang, S.; and Chen, S. 2023. KD-Crowd: A Knowledge Distillation Framework for Learning from Crowds. *Frontiers of Computer Science*.

Liang, J.; Homayounfar, N.; Ma, W.-C.; Xiong, Y.; Hu, R.; and Urtasun, R. 2020. Polytransform: Deep polygon transformer for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9131–9140.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, 10012–10022.

Mishkin, D.; and Matas, J. 2015. All you need is a good init. *arXiv preprint arXiv:1511.06422*.

Oquab, M.; Darcet, T.; Moutakanni, T.; Vo, H.; Szafraniec, M.; Khalidov, V.; Fernandez, P.; Haziza, D.; Massa, F.; El-Nouby, A.; et al. 2023. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*.

Qin, R.; Zhang, G.; and Tang, Y. 2023. On the Transferability of Learning Models for Semantic Segmentation for Remote Sensing Data. *arXiv preprint arXiv:2310.10490*.

Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, 8748–8763. PMLR.

Ren, S.; Wei, F.; Zhang, Z.; and Hu, H. 2023. TinyMIM: An empirical study of distilling MIM pre-trained models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3687–3697.

Sun, Q.; Liu, Y.; Chua, T.-S.; and Schiele, B. 2019. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 403–412.

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.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, volume 29, 3630–3638.

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

Wang, Q.-F.; Geng, X.; Lin, S.-X.; Xia, S.-Y.; Qi, L.; and Xu, N. 2022a. Learngene: From Open-World to Your Learning Task. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 8557–8565.

Wang, W.; Bao, H.; Dong, L.; Bjorck, J.; Peng, Z.; Liu, Q.; Aggarwal, K.; Mohammed, O. K.; Singhal, S.; Som, S.; et al. 2022b. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. *arXiv preprint arXiv:2208.10442*.

Wang, W.; Wei, F.; Dong, L.; Bao, H.; Yang, N.; and Zhou, M. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33: 5776–5788.

Wu, K.; Zhang, J.; Peng, H.; Liu, M.; Xiao, B.; Fu, J.; and Yuan, L. 2022. Tinyvit: Fast pretraining distillation for small vision transformers. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXI*, 68–85. Springer.

Zhang, C.; Song, N.; Lin, G.; Zheng, Y.; Pan, P.; and Xu, Y. 2021. Few-shot incremental learning with continually evolved classifiers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 12455–12464.

Zhang, J.; Peng, H.; Wu, K.; Liu, M.; Xiao, B.; Fu, J.; and Yuan, L. 2022. Minivit: Compressing vision transformers with weight multiplexing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12145–12154.

Zhang, L.; Bao, C.; and Ma, K. 2021. Self-distillation: Towards efficient and compact neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8): 4388–4403.

Zhou, B.; Cui, Q.; Wei, X.-S.; and Chen, Z.-M. 2020. Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9719–9728.