Abstract

Recently, vision Transformers (ViTs) are developing rapidly and starting to challenge the domination of convolutional neural networks (CNNs) in the realm of computer vision (CV). With the general-purpose Transformer architecture for replacing the hard-coded inductive biases of convolution, ViTs have surpassed CNNs, especially in data-sufficient circumstances. However, ViTs are prone to over-fit on small datasets and thus rely on large-scale pre-training, which expends enormous time. In this paper, we strive to liberate ViTs from pre-training by introducing CNNs’ inductive biases back to ViTs while preserving their network architectures for higher upper bound and setting up more suitable optimization objectives. To begin with, an agent CNN is designed based on the given ViT with inductive biases. Then a bootstrapping training algorithm is proposed to jointly optimize the agent and ViT with weight sharing, during which the ViT learns inductive biases from the intermediate features of the agent. Extensive experiments on CIFAR-10/100 and ImageNet-1k with limited training data have shown encouraging results that the inductive biases help ViTs converge significantly faster and outperform conventional CNNs with even fewer parameters.

1. Introduction

The great successes of the convolutional neural networks (CNNs) [21, 23, 29, 42] have liberated researchers from handcrafting visual features [14, 33]. By means of the inductive biases [10], i.e., focusing on the localized features and weight sharing, CNNs are potent tools for tackling visual recognition tasks [5, 21, 36]. Nevertheless, such biases have constrained their abilities towards building deeper and larger models, as they have ignored the long-range dependencies [15, 18].

In recent years, Transformers [46] have been proposed for replacing inductive biases with a general-purpose network architecture in natural language processing (NLP). Exclusively relying on multi-head attention mechanisms (MHA), Transformers have the inborn capability to capture the global dependencies within language tokens and have become the de facto preferred data-driven models in NLP [2, 17, 35]. Inspired by this, a growing number of researchers have introduced the Transformer architecture into the realm of computer vision (CV) [4, 18, 45, 56]. It turns out an encouraging discovery that vision Transformers (ViTs) outperform state-of-the-art (SOTA) CNNs by a large margin with a similar amount of parameters.

Despite the appealing achievements, ViTs suffer from poor performance, especially without adequate annotations or strong data augmentation strategies [8, 18, 45]. The reasons for this circumstance are two folds: on the one hand, the majorly adopted multi-head self-attention mechanisms (MHSA) in ViTs have dense connections against convolution [11], which is hard to optimize without prior knowledge; on the other hand, Chen et al. [8] have illustrated that ViTs tend to converge to minima with sharp regions, usually related to limited generalization capability and overfitting problems [7, 25]. However, the typical training scheme of Transformers in NLP [2, 17] relies on the large-scale pre-training and then fine-tuning for downstream tasks, which consume enormous GPU (TPU) time and energy [4, 18, 45]. For instance, Dosovitskiy et al. [18] spend thousands of TPU days to pre-train a ViT with 303M images. Spontaneously, it raises the following question: how can we optimize ViTs efficiently without pre-training.
To the best of our knowledge, existing approaches focused on the problem can be mainly divided into two parts. The first line of approaches attempts to bring inductive biases back into Transformers, such as sparse attention [6, 12, 27] and token aggregation [52]. Such heuristic modifications to ViTs will inevitably lead to the sophisticated tuning of plenty of hyperparameters. The second line of approaches [8, 24, 45] aims at constructing suitable training schemes for Transformers, which helps them converge with better generalization ability. In particular, Chen et al. [8] utilize the sharpness-aware minimizer (SAM) [20] to find smooth minima, while [24, 45] optimize a Transformer by distilling knowledge from a pre-trained teacher. Nonetheless, these methods still require pre-training on mid-sized datasets, such as ImageNet-1k [29], and how to efficiently train ViTs with relatively small datasets from scratch remains an open question.

Motivated by the distillation approaches [22, 24, 45] that utilize a teacher model to guide the optimization direction of the student, in this paper, we strive to make one step further towards optimizing ViTs with the help of an agent CNN, which also learns from scratch along with the ViT. Our goal is to inject the inductive biases from the agent CNN into the ViT without modifying its architecture and design a more friendly optimization process so that ViTs can be customized on small-scale datasets without pre-training.

To this end, we propose a novel optimization strategy for training vision Transformers in the bootstrapping form so that even without pre-training on mid-sized datasets or strong data augmentations, ViTs can still be competitive when lack of training data. Specifically, as shown in Fig. 1, we first propose an agent CNN designed corresponding to the given ViT, and with the inductive biases, the agent will converge faster than the ViT. Then we jointly optimize the ViT along with the agent in the mutual learning framework [55], where the intermediate features of the agent supervise the ViT with the inductive biases for fast convergence. In order to reduce the training burden, we further share the parameters of the ViT to the agent and propose a bootstrapping learning algorithm to update the shared parameters. We have conducted extensive experiments on CIFAR-10/100 datasets [28] and ImageNet-1k [29] under the lack-of-data settings. Experimental results demonstrate that: (1) our method has successfully injected the inductive biases to ViTs as they converge significantly faster than training from scratch and eventually surpass both the agents and SOTA CNNs; (2) the bootstrapping learning method can efficiently optimize the shared weights without the extra set of parameters.

Our contributions are summarized as three folds:

1. We propose agent CNNs constructed based on standard ViTs, for training ViTs efficiently with shared weights and inductive biases.
2. We propose a novel bootstrapping optimization algorithm to optimize the shared parameters.
3. Our experiments show that ViTs can outperform the SOTA CNNs even without pre-training by adopting both inductive biases and suitable optimizing goals.

2. Related Work

2.1. Vision Transformers

With the powerful self-attention mechanisms, the Transformer [46] has been the SOTA and preferred model in NLP [2, 3, 17]. Inspired by the impressive success of Transformers in NLP, researchers are starting to introduce Transformers for tackling CV tasks. ViT [18] is a groundbreaking work that utilizes the pure Transformer architecture for image classification and has achieved great successes. The variants of ViTs [9, 31, 45, 48, 52, 57] are further utilized for more complex CV tasks, e.g., semantic segmentation [40, 56] and object detection [4, 13]. However, the ViTs rely on large-scale pre-training and have shown poor performance with limited training data. To address this problem, some approaches try to introduce inductive biases to ViTs with heuristic modification, e.g., sparse attention [6, 12, 27], token aggregation [52]. The others aim to propose novel training schemes tailored for Transformers [8, 24, 45]. Nonetheless, these methods still require pre-training on mid-scale datasets, such as ImageNet-1k, and it still remains an open question of how to optimize ViTs efficiently without pre-training, especially on small-scale datasets. To resolve this problem, we strive to inject CNNs’ inductive biases into ViTs without modifications to the network architecture.

2.2. Knowledge Distillation

Knowledge distillation (KD) [22] is an effective model compression technic which the hidden knowledge of the teacher is transferred to the student by supervising with soft labels. To sufficiently transfer the knowledge, FitNets [37] additionally use intermediate features for supervision, and the following works [39, 44, 53] extract the deeper level of information in different aspects. More recently, mutual learning [55], a variant of KD, has attracted many interests as all the networks are students and simultaneously learn from each other. This practical learning strategy has been applied to person re-identification [19, 47], object detection [49], and face recognition [1, 16]. Apart from the applications of mutual learning, some researchers focus on improving mutual learning by introducing more supervision like the intermediate features [50] or feature fusion [26]. Inspired by this, we propose utilizing an agent CNN optimized jointly with the ViT. The hard-coded inductive biases are transferred to the ViT under the mutual learning framework with adaptive intermediate feature supervision.
3. Method

In this section, we first introduce the preliminaries of CNNs and ViTs. Then, based on the relationship of convolution layers and MHSA layers, agent CNNs are proposed to help train ViTs. Finally, we delineate the bootstrapping optimization algorithm where the agent and ViT are jointly optimized without pre-training. The workflow of our method is demonstrated in Fig. 2.

3.1. Preliminaries

3.1.1 Convolution

Convolution is the central part of CNNs, which accepts a two-dimensional feature map. For the sake of future discussion, we formalize the forward propagation with a sequence of visual tokens $X = (x_1, \ldots, x_n) \in \mathbb{R}^{n \times d_{in}}$ as its input, each of which is a $d_{in}$-dimensional embedded vector. Thus, the output sequence of convolution with kernel size $(k_h, k_w)$ is the sum of linear projection of $X$:

$$Y_C = \sum_{i=1}^{N} \Phi_i X W_i,$$

where $\Phi_i$ is the sparse matrix representing the hard-coded inductive biases of localized dependencies, the size of receptive field $N = k_h \times k_w$, and the projection matrix $W_i \in \mathbb{R}^{d_{in} \times d_{out}}$ is trainable.

It is worth noting that the $1 \times 1$ convolution has the form of $Y = XW$, which is equivalent to a fully connected (FC) layer with the same projection matrix $W$.

3.1.2 MHSA

The multi-head self-attention mechanism (MHSA) in ViTs takes a sequence of visual tokens as its input and can also be formalized similar to Eq. (1):

$$Y_M = \sum_{h=1}^{H} \Psi_h X W_h^{VO},$$

where $H$ is the number of heads, $W_h^{VO} = W_h^V W_h^O$ is the combination of two projection matrices ($W_h^V \in \mathbb{R}^{d \times d_k}$ and $W_h^O \in \mathbb{R}^{d_k \times d}$, $d = d_k$ is the embedding dimension), and $\Psi_h \in \mathbb{R}^{n \times n}$ is the dense matrix representing all the token relationships. Particularly,

$$\Psi_h = \sigma \left( \frac{X W_h^Q W_h^K X^\top}{\sqrt{d_k}} \right)$$

is computed based on the similarity of linearly projected $X$ by $W_h^Q$ and $W_h^K \in \mathbb{R}^{d \times d_k}$ respectively, where $\sigma(\cdot)$ denotes the softmax operation to each row of the input matrix.

3.2. Agent CNN

Inspired by the similarity of Eq. (1) and Eq. (2) that the convolution layer can be treated as a special case of MHSA layer with sparse relationship matrices $\Psi$, we propose constructing an agent CNN based on a given ViT that will converge faster when trained from scratch.

3.2.1 Generalized Convolution

To begin with, we propose a generalized convolution layer in which the size of its receptive field equals the head number $H$ of a MHSA layer, named as CONV, with hard-coded...
inductive biases \( \{ \tilde{\Phi}_h \}_{h=1}^H \):

\[
Y_{\text{CONV}} = \sum_{h=1}^H \tilde{\Phi}_h X W_h,
\]

where \( \tilde{\Phi}_1, \ldots, \tilde{\Phi}_H \in \mathbb{R}^{n \times n} \) are extracted from the set \( \{ \Phi_1, \ldots, \Phi_N \} \) generated by the \( \sqrt{H} \times \sqrt{H} \) convolution \((N' = \lceil \sqrt{H} \rceil^2)\). When the head number \( H \) is a perfect square number, i.e., \( H = K^2, K > 0 \), the generalized convolution layer will be equivalent to the \( K \times K \) convolution layer.

### 3.2.2 Constructing Agent CNN

We start with a standard ViT model (in Fig. 3a) with \( m \) encoder layers and finally build an agent CNN for introducing the inductive biases of CNNs. Fig. 3b illustrates a base agent CNN by simply replacing the MHSAs of ViT with CONV layers, which has introduced sparsity and localized biases. Besides, the FFNs in the agent are composed of two \( 1 \times 1 \) convolution layers which are equivalent to the fully connected layers in the FFNs of Transformers.

Moreover, as many preferred CNNs share the feature pyramid architecture \([21, 30, 38, 54]\) that the spacial size of feature maps shrunk as going deeper, we construct the final res-like agent CNN (in Fig. 3c) by: (1) introducing a ResNet-style input projection block which contains two convolution layers and one max-pooling layer, which strides are 2, (2) adopting a configurable down-sampling module after each encoder layer.

With the hard-coded inductive biases, the agent can converge faster and with higher performance than training the corresponding ViT from scratch as shown in Fig. 5c.

### Weight Sharing

Utilizing the homologous network architecture, our proposed agent accepts shared weights from the ViT model to reduce the training burden. Due to the equivalence of \( 1 \times 1 \) convolution and FC layers, the FFNs in each encoder block of ViT can be directly shared by the agent. Furthermore, shared with the output projection matrix \( W_h^{VO} \) of MHSAs in Eq. (2), the CONV has the form of

\[
Y_{\text{CONV}} = \sum_{h=1}^H \tilde{\Phi}_h X W_h^{VO}.
\]

Let \( y_c \) and \( \tilde{y}_c \) be the \( c \)-th token of the output of MHSAs and shared CONV accordingly. With the assumption that the input sequences are the same, denoted by \( X \), the difference \( y_{err} = y_c - \tilde{y}_c \) can be written as

\[
y_{err} = \sum_{h=1}^H (\tilde{\Phi}_h - \Psi_h) X W_h^{VO},
\]

where \( \tilde{\Phi}_h \) and \( \Psi_h \) are the \( c \)-th row of matrix \( \tilde{\Phi}_i \) and \( \Psi_i \) respectively. As there are no more than one non-zero element in \( \tilde{\Phi}_h \) (proved in Appendix A), we can minimize the magnitude of \( y_{err} \) by learning sparse and localized dependencies of matrix \( \Psi_h \), parameterized by \( W^{Q}_h \) and \( W^{K}_h \) in Eq. (3).

### 3.3. Bootstrapping Optimization

In this section, we describe how to jointly optimize the ViT and agent by introducing the optimization objective and training strategy.

#### 3.3.1 Adaptive Intermediate Supervision

To inject the inductive biases of the agent into ViT without modifying ViT’s architecture, we propose the adaptive intermediate supervision where the adapted feature maps of the agent supervise the corresponding visual sequences of the ViT. Let \( F_{A}^{(\ell)} \) and \( F_{V}^{(\ell)} \) denote the flattened feature maps and visual sequence of the \( \ell \)-th encoder layer of the agent and ViT in respective. The adaptive intermediate loss of \( \ell \)-th layer of the ViT and agent is defined as

\[
L_{\text{feat}}^{(\ell)} = \left\| F_{A}^{(\ell)} - \hat{F}_{A}^{(\ell)} \right\|_{2}^{2} + \left\| F_{V}^{(\ell)} - \hat{F}_{V}^{(\ell)} \right\|_{2}^{2},
\]

where \( \hat{F} := \text{Adapt}(F) \) is the adapted feature, obtained from sequence interpolation or 2-dimensional average pooling. We have compared the different adaptive approaches in Sec. 4.3.

Finally, the adaptive intermediate supervision is the sum of all assigned layers \( \Lambda \):

\[
L_{\text{feat}} = \sum_{\ell \in \Lambda} L_{\text{feat}}^{(\ell)}.
\]
Weight Sharing learning will degenerate to multi-task learning. As shown in Fig. 4, the input of multi-task model is only one, while the input of the bootstrapping model is more than one. In our case, as we constrain the difference between inputs of each encoder layer by Eq. (8) and when the norm of $y_{	ext{en}}$ is small enough, the bootstrapping learning will degenerate to multi-task learning.

### 3.3.2 Optimization Objective

Except for intermediate supervision, we introduce the mutual learning framework [55] that the predicted probabilities by the ViT (denoted as $p_V$) and the agent (denoted as $p_A$) learn from each other as

$$L_{\text{mutual}} = L_{\text{KD}}(p_V, \bar{p}_A, y; T) + L_{\text{KD}}(p_A, \bar{p}_V, y; T),$$  

where $\bar{p}$ represents that the variable $p$ is treated as a constant vector, i.e., no gradient is computed with regard to the variables in the forward propagation paths, $L_{\text{KD}}$ is the knowledge distillation loss defined in [22] with temperature $T$, and $y$ denotes the ground-truth label of the input image.

Above all, the optimization objective is summarized as

$$L = \alpha L_{\text{feat}} + \beta L_{\text{mutual}},$$  

and $\alpha$ and $\beta$ are the weighting hyperparameters for balancing the two terms.

### 3.3.3 Bootstrapping Training Algorithm

The bootstrapping training algorithm is given in Algorithm 1 where gradients computed from each network are aligned and jointly update the shared weights. The gradient alignment function $\text{Align} (\nabla_S \beta, \nabla_A \beta)$ modifies the negative gradient direction from the agent as [51].

Relationship with multi-task learning. It is worth pointing out that bootstrapping learning is slightly different from multi-task learning. As shown in Fig. 4, the input of multi-task model is only one, while the input of the bootstrapping model is more than one. In our case, as we constrain the difference between inputs of each encoder layer by Eq. (8) and when the norm of $y_{\text{en}}$ is small enough, the bootstrapping learning will degenerate to multi-task learning.

#### Algorithm 1 Bootstrapping optimizer for training the shared weights in FFN and MHSAs layers in the ViT.

**Input:** $\Theta_S$: the set of shared weights; $\Theta_V, \Theta_A$: the set of private weights in the ViT and agent respectively; $E_V^{(l)}(\cdot; \Theta_S, \Theta_V)$, $E_A^{(l)}(\cdot; \Theta_S, \Theta_A)$: the $l$-th encoder layer of the ViT and agent respectively; $\lambda$: learning rate.

1. while not converge do
2. compute the input feature map (sequence) $X_V, X_A$ to the $l$-th encoder layer from the input image
3. $Y_V \leftarrow E_V^{(l)}(X_V; \Theta_S, \Theta_V)$
4. $Y_A \leftarrow E_A^{(l)}(X_A; \Theta_S, \Theta_A)$
5. Compute the gradients $\nabla_{\beta} V, \nabla_{\beta} A$ w.r.t. $\Theta_S$ and $\Theta_V$ respectively ($\Theta_A$ keep as constant variables).
6. Compute the gradients $\nabla_{\beta} S, \nabla_{\beta} A$ w.r.t. $\Theta_S$ and $\Theta_A$ respectively ($\Theta_V$ keep as constant variables).
7. $\Theta_V \leftarrow \Theta_V - \lambda \nabla_{\beta} V$, $\Theta_A \leftarrow \Theta_A - \lambda \nabla_{\beta} A$ # update private weights
8. $\Theta_S \leftarrow \Theta_S - \lambda/2 (\nabla_{\beta} S + \text{Align}(\nabla_{\beta} S, \nabla_{\beta} V))$ # update shared weights in bootstrapping way
9. end while

#### 4. Experiments

##### 4.1. Implementation

**Datasets.** Three widely-used image classification datasets are adopted to evaluate our proposed method as benchmarks, including CIFAR-10 [28], CIFAR-100 [28], and ImageNet-1k [29]. In particular, to simulate lack-of-data circumstances, 1%, 5%, and 10% labeled samples are randomly extracted from the training partition of the ImageNet dataset. Even though ViTs require strong data augmentations in previous approaches [18, 45, 48, 52], in our implementation, both CNNs and ViTs are only optimized with several simple augmentation methods, such as random resized cropping and random horizontal flipping.

**Vision Transformers.** We follow the network architectures introduced in [18] and [45] with slight modifications to the head number and embedding dimensions. The detailed settings of ViTs are presented in Tab. 2, in which ViT-S is a relatively small model with 6 layers, and the ViT-B is the same as DeiT [45] with 12 layers.

**Agent CNNs.** The agents are constructed according to the given ViTs and thus share the same network settings as ViTs (shown in Tab. 2). Besides, detailed network architectures of the base and res-like agents are described in Appendix B, e.g., input projection and down-sampling layers.

**Training Details and Selection of Hyperparameters.** We implement our method using the Pytorch [34] frame-
Table 1. Comparison results on CIFAR-10 and CIFAR-100. The top-1 accuracy, number of parameters, and FLOPs are reported separately. ‘†’ indicates that the initial weights of pre-trained ViT-B are acquired from the official repository of DeiT. The comparison settings are classified in the ‘Model’ column. The values in blue color indicates top-1 accuracy improvements compared with the corresponding ViT trained from scratch.

| Model   | Method     | CIFAR-10     | CIFAR-100    |
|---------|------------|--------------|--------------|
|         |            | Acc  | #param. | FLOPs | Acc   | #param. | FLOPs  |
| CNNs    | EfficientNet-B2 | 94.14 | 7.71M | 0.70G | 75.55 | 7.84M | 0.70G |
|         | ResNet50   | 94.92 | 23.53M | 4.14G | 77.57 | 23.71M | 4.14G |
|         | Agent-S    | 94.18 | 8.66M | 3.37G | 74.62 | 8.73M | 3.37G |
|         | Agent-B    | 94.83 | 25.05M | 9.46G | 74.78 | 25.91M | 9.46G |
| ViTs    | ViT-S      | 87.32 | 6.28M | 1.37G | 61.25 | 6.30M | 1.37G |
|         | ViT-S-SAM  | 87.77 | 6.28M | 1.37G | 62.60 | 6.30M | 1.37G |
|         | ViT-S-Sparse | 87.43 | 6.28M | 1.37G | 62.29 | 6.30M | 1.37G |
|         | ViT-B      | 79.24 | 21.67M | 4.62G | 53.07 | 21.70M | 4.62G |
|         | ViT-B-SAM  | 86.57 | 21.67M | 4.62G | 58.18 | 21.70M | 4.62G |
|         | ViT-B-Sparse | 83.87 | 21.67M | 4.62G | 57.22 | 21.70M | 4.62G |
| Pre-trained ViTs | ViT-S | 95.70 | 6.28M | 1.37G | 80.91 | 6.30M | 1.37G |
|         | ViT-B†     | 97.17 | 21.67M | 4.62G | 84.95 | 21.70M | 4.62G |
| Ours Joint | Agent-S     | 94.90 | 8.66M | 3.37G | 74.06 | 8.73M | 3.37G |
|         | ViT-S      | 95.14 (+7.82) | 6.28M | 1.37G | 76.19 (+14.94) | 6.30M | 1.37G |
|         | Agent-B    | 95.06 | 25.05M | 9.46G | 76.57 | 25.91M | 9.46G |
|         | ViT-B      | 95.00 (+15.76) | 21.67M | 4.62G | 77.83 (+24.76) | 21.70M | 4.62G |
| Ours Shared | Agent-S | 93.22 | - | - | 74.06 | - | - |
|         | ViT-S      | 93.72 (+6.40) | 6.28M | 1.37G | 75.50 (+14.25) | 6.30M | 1.37G |
|         | Agent-B    | 92.66 | - | - | 74.11 | - | - |
|         | ViT-B      | 93.34 (+14.10) | 21.67M | 4.62G | 75.71 (+22.64) | 21.70M | 4.62G |

Table 2. Detailed network architecture settings of ViTs and agent CNNs.

| ViT-S | ViT-B | Agent-S | Agent-B |
|-------|-------|---------|---------|
| Layers \((m)\) | 6 | 12 | 6 | 12 |
| Hidden Size \((d)\) | 288 | 384 | 288 | 384 |
| Heads \((H)\) | 9 | 6 | 9 | 9 |

AdamW [32] is used to optimize ViTs and agent CNNs in both standalone and joint training schemes, with the learning rate of \(10^{-3}\) and a weight decay of \(5 \times 10^{-2}\). However, conventional CNNs, such as ResNet [21] and EfficientNet [43], are optimized by SGD [41] with a learning rate of \(5 \times 10^{-2}\) and weight decay of \(5 \times 10^{-4}\). We train all the settings for 240 epochs with the batch size of 32 on two Nvidia Tesla A100 GPUs and use cosine annealing as the learning rate decay schedule.

There are several hyperparameters involved in our method, including \(\alpha\) and \(\beta\) in Eq. (10) and the temperature \(T\) for knowledge distillation loss in Eq. (9). We set \(\alpha = 1\), \(\beta = 10\) and \(T = 4\) as default values and the sensitivity analysis is described in Sec. 4.5.

Finally, we set the intermediate feature supervisions decay linearly for preserving the capacity of the ViTs, and different decay strategies are compared in Sec. 4.3.

### 4.2. Experimental Results

We evaluate our proposed method on CIFAR and ImageNet datasets and set up the following comparison settings:

- **CNNs**: the standalone agents and traditional CNNs, e.g., EfficientNet and ResNet.
- **ViTs**: the original ViTs and variants for training with high efficiency, e.g., ViT-Sparse [12] and ViT-SAM [8]. All of them are trained from scratch.
- **Pre-trained ViTs**: vision Transformers which have been pre-trained on ImageNet-1k and then fine-tuned to CIFAR datasets.

---

\(^2\)Our code will be available soon.
**Performance on CIFAR Datasets.** The comparison results on CIFAR-10/100 are shown in Tab. 1, where the top-1 accuracy, number of parameters, and FLOPs are presented for each setting. The discoveries are itemized as follows.

1. When the agent CNNs are trained alone, the performance can hardly surpass the traditional CNNs, even if most of the inductive biases are hard-coded into the agents.

2. Without pre-training or strong data augmentations, the ViTs perform terribly due to the dense connection of MHSA layers. Although Chen et al. [8] have shown that ViTs can be optimized with SAM optimizer, librated from the pre-training on mid-scale datasets, such as ImageNet-1k, it may not be the best choice for small datasets like CIFAR.

3. Our proposed method has significantly outperformed the baseline settings, including the original ViTs and the variations. Particularly, ViT-S surpasses the original baseline by 7.82% on CIFAR-10 and 14.49% on CIFAR-100, which outperforms the agent and EfficientNet-B2 with fewer parameters.

4. In the ‘Shared’ settings, the bootstrapping learning strategy has shown that the shared weights can be optimized robustly with a limited deterioration of accuracy. Such results are encouraging that the weights of ViTs can be directly transferred to a framework of hard-coded inductive biases so that ViTs can utilize the inductive biases without an extra set of parameters or sophisticated modifications.

Moreover, we plot the accuracy learning curves of our method along with the baseline settings in Fig. 5. It clearly illustrates that ViTs can converge as fast as CNNs and finally achieve the higher upper bound than CNNs’.

**4.3 Ablation Study**

In this section, we go deep into our proposed method to figure out the function of loss terms (introduced in Sec. 3.3.2), the effects of different decay strategies (in Sec. 4.1), and adaptive functions (in Sec. 3.3.1).
Table 4. Ablation Study of the joint learning ViT-S method on CIFAR-100. In the column of ‘Feat’, '✓' denotes using the default settings, ‘No Decay’ represents the weight $\beta$ of the $L_{\text{feat}}$ keep constant throughout the whole training process, and ‘AP-2D’ means using 2D average pooling as adaptive function in Eq. (7).

| Settings | Mutual | Feat | Acc  | Training Time |
|----------|--------|------|------|---------------|
| Baseline | ×      | ×    | 61.25| 3.3h          |
| KD Only  | ✓      | ×    | 67.16| 3.6h          |
| Feat Only| ×      | No Decay | 73.59| 3.8h          |
|          | ✓      | AP-2D | 71.06| 6.6h          |
| Both     | ✓      | No Decay | 75.15| 3.8h          |
|          | ✓      | 76.19 | 3.8h  |

4.3.1 Ablation of Loss Terms

As delineated in Sec. 3.3.2, the ultimate optimization objective has two terms: the adaptive intermediate supervision $L_{\text{feat}}$ and the mutual learning term $L_{\text{mutual}}$. We evaluate the joint learning settings when supervised with the two terms separately in Tab. 4, showing that both loss terms have contributed to the final result. Particularly, $L_{\text{feat}}$ boosts the accuracy by 13.69 percentage points. Additionally, learning curves are plotted in Fig. 5c for better illustration. We can observe that the ‘Feat’ term produces a more competitive result as it converges significantly faster than using the ‘Mutual’ term. Therefore, the supervision through the intermediate features has successfully injected the inductive biases into the ViT.

4.3.2 Ablation of Decay Strategy

The influence of the feature supervision decay strategy is shown in Tab. 4. Without the decay strategy, the performance has declined by 1.35%. This can be explained that the inductive biases constrain the further performance of ViTs as the agents cannot capture the long-range dependencies.

4.3.3 Ablation of Adaptive Functions

We evaluate our method with two intermediate feature adaptive functions: 1D sequence interpolation (by default) and 2D average pooling. The comparison results are shown in Tab. 4, in which the sequence interpolation is superior to the average pooling.

4.4. The Choice of Agent CNNs

In Sec. 3.2.2, we have introduced agent CNNs with two different network architectures (base and res-like). Tab. 5 shows that the performances with the res-like configuration are uniformly better than the base agent.

4.5. Sensitivity Analysis of Hyperparameters

The sensitivity analysis of the hyperparameters in our methods is illustrated in Fig. 6, including $\alpha$, $\beta$ in Eq. (10), and $T$ in Eq. (9). It shows that our method is robust to the variation of $\alpha$ and $T$. However, $\beta$ has a more significant influence, as the ViTs perform better when more inductive biases are used to supervise the ViTs.

5. Conclusion and Future Work

In this paper, we propose to dissolve optimizing vision Transformers (ViTs) efficiently without pre-training or strong data augmentations. Our goal is to introduce inductive biases from convolutional neural networks (CNNs) to ViTs while preserving the network architecture of ViTs for higher upper bound and further setting up more suitable optimization objectives. To this end, we propose to optimize the ViT jointly with an agent CNN constructed corresponding to the network architecture of the ViT. The ViT learns inductive biases through adaptive intermediate supervision and predicted probabilities. Besides, a bootstrapping training algorithm is proposed to optimize both the ViT and agent with weight sharing. Extensive experiments have shown encouraging results that the inductive biases help ViTs converge significantly faster and outperform conventional CNNs with fewer parameters. In future work, we will strive to supervise ViT with more interpretability features which will further improve the generalization ability of ViTs.

Table 5. Joint training results on CIFAR-100 when using the agent with different network architectures.

|            | Agent-S | ViT-S-Joint | Agent-S-Joint |
|------------|---------|-------------|--------------|
| Base       | 72.73   | 73.18       | 73.79        |
| Res-like   | 74.78   | 76.19       | 74.06        |

Figure 6. Sensitivity analysis of hyperparameters in our method.
