Learning Scalable Model Soup on a Single GPU: An Efficient Subspace Training Strategy

Tao Li[1,*], Weisen Jiang[2,4,*], Fanghui Liu[3], Xiaolin Huang[(✉)1], James T. Kwok[2]

[1] Department of Automation, Shanghai Jiao Tong University, China
[2] Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong
[3] Department of Computer Science, University of Warwick, United Kingdom
[4] Department of Computer Science and Engineering, Southern University of Science and Technology, China
{li.tao,xiaolinhuang}@sjtu.edu.cn, waysonkong@gmail.com
fanghui.liu@warwick.ac.uk, jamesk@cse.ust.hk

Abstract. Pre-training followed by fine-tuning is widely adopted among practitioners. The performance can be improved by "model soups" [47] via exploring various hyperparameter configurations. The Learned-Soup, a variant of model soups, significantly improves the performance but suffers from substantial memory and time costs due to the requirements of (i) having to load all fine-tuned models simultaneously, and (ii) a large computational graph encompassing all fine-tuned models. In this paper, we propose **M**emory **E**fficient **H**yperplane **L**earned Soup (MEHL-Soup) to tackle this issue by formulating the learned soup as a hyperplane optimization problem and introducing block coordinate gradient descent to learn the mixing coefficients. At each iteration, MEHL-Soup only needs to load a few fine-tuned models and build a computational graph with one combined model. We further extend MEHL-Soup to MEHL-Soup+ in a layer-wise manner. Experimental results on various ViT models and data sets show that MEHL-Soup(+) outperforms Learned-Soup(+) in terms of test accuracy, and also reduces memory usage by more than $13\times$. Moreover, MEHL-Soup(+) can be run on a single GPU and achieves $9\times$ speed up in soup construction compared with the Learned-Soup. The code is released at `https://github.com/nblt/MEHL-Soup`.

Keywords: Model Soups · Weight Averaging · Subspace Optimization

1 Introduction

Pre-training followed by fine-tuning is a widely adopted training pipeline for deep neural networks [8, 19, 20, 50, 51]. Typically, one starts with a large model pre-trained on an extensive collection of datasets and then fine-tunes multiple models with various hyperparameter configurations to seek better performance. To maximize the benefits of these fine-tuned models, the concept of "*model soups*" [47]

---

[*] Equal contribution. Work done when Tao was a visiting student at HKUST.

has been introduced to selectively average the weights of these models for an improved souping model while keeping the inference efficiency as a single model. Model soups have achieved significant success and widely used in various domains, such as out-of-distribution performance [4, 33, 47], reinforcement learning [34], model pruning [49, 53], and adversarial robustness [5, 11].

There are two representative categories of model soups: (i) *greedy soup* [4, 33, 47], in which fine-tuned models are added to the soup sequentially in a greedy order; and (ii) *learned soup* [23, 47], in which models are mixed by coefficients learned from a validation set. Greedy soup is simple and effective, but may not explore the full potential of all fine-tuned models as the models in the soup are equally averaged while others are discarded [43]. The learned soup, which learns the soup's mixing coefficients via gradient-based optimization on the validation set, is more general and achieves better performance (e.g., test accuracy) in practice [11, 47]. However, the learned soup suffers from a heavy burden in computation and memory since it requires loading *all* models into memory simultaneously and building a computational graph on *all* models. For example, Learned-Soup requires more than 200GB of memory for averaging 72 fine-tuned ViT-B/32 models [32], and thus the training process has to be conducted in CPU memory [47], which is time-consuming. Hence, Learned-Soup is inefficient in both memory and computation, hindering its application to large models.

In this paper, we develop a scalable and efficient approach to learning the model soup. It works well under limited computational resources, even on a single GPU. We formulate the learned soup as a subspace learning problem and propose a hyperplane optimization objective, which only requires a computational graph on the combined model. Furthermore, we introduce block coordinate gradient descent [30, 45, 48] to optimize the mixing coefficients, where only a mini-batch of models needs to be loaded into memory at each iteration. This not only scales well but also achieves better performance as the introduced stochasticity benefits generalization [2, 21, 42].

The proposed **M**emory-**E**fficient **H**yperplane **L**earned Soup (MEHL-Soup) maintains memory and time efficiency while benefiting significant performance improvement from trainable coefficients. Furthermore, it is extended in a layer-wise manner (MEHL-Soup+) for boosting performance. To be specific, our main contributions can be summarized as follows:

- We propose MEHL-Soup(+), a computation- and memory-efficient approach to learning the mixing coefficients of model soup based on a novel hyperplane optimization objective, which allows for learning extrapolated coefficients.
- We adopt block coordinate gradient descent to enable training of the model soup scalable and memory-efficient, which can be run on a single GPU. Convergence of MEHL-Soup(+) is also established.
- Experimental results show that MEHL-Soup(+) brings $13\times$ reduction in memory and $9\times$ in soup construction time compared with Learned-Soup(+) along with consistently better performance. Moreover, our findings reveal that compared to Greedy-Soup, MEHL-Soup(+) substantially reduces the

cost of fine-tuning and exhibits lower sensitivity to top-performing fine-tuned models, making it more preferable in practice.

2   Related Work

Weight Averaging is a widely used technique in deep learning and optimization for improving generalization [15,33,44,47] and convergence [12,18,23,29,40]. Along the same training trajectory, Izmailov et al. [15] show that averaging the weights at the latter stage of training accompanied with a constant learning rate schedule can significantly improve generalization. Kaddour et al. [18] introduce the latest weight averaging (LAWA) to accelerate the convergence of training. In the context of combining weights from different training trajectories, Wortsman et al. [47] propose to selectively combine multiple fine-tuned models in the parameter space to boost performance. The combined model is also called a *model soup* [47]. In inference, only the model soup needs to be deployed and served. Thus, it is computation- and memory-efficient compared with serving all fine-tuned models to combine models in the output space (i.e., model ensemble [52]). Model averaging has achieved promising performance in a wide variety of applications, including federated learning [3], robust training [1,35], and multi-task training [13,28,31], open-vocabulary recognition [14], and language models alignment [34].

Subspace Training. Recent studies [9,22,24,26,46] show that neural networks can be learned in a tiny subspace. The subspace can be constructed from random basis [9,22], training dynamics [24], models fine-tuned from a pre-trained model [47], and multiple task-specific models [16]. Existing subspace training methods require loading all models into the memory and constructing a computational graph on all of them, leading to the scalability issue for large models. In contrast, the training strategy proposed in this work only needs a computational graph on the combined model and loads a mini-batch of models.

Preliminary. Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a compact metric space and $\mathcal{Y} \subseteq \mathbb{N}$ be the label space for classification. The training data $\mathcal{D}^{\mathrm{tr}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and validation data $\mathcal{D}^{\mathrm{vl}}$ are drawn from an unknown probability distribution on $\mathcal{X} \times \mathcal{Y}$. We aim at seeking a hypothesis (i.e., deep network in this work) $f : \mathcal{X} \to \mathcal{Y}$ such that $f(\mathbf{x}, \boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$, is a good approximation of the label y corresponding to a new sample $\mathbf{x} \in \mathcal{X}$. The loss function $\ell(f(\mathbf{x}, \boldsymbol{\theta}), y)$ (e.g., cross-entropy loss) measures the discrepancy between the prediction $f(\mathbf{x}, \boldsymbol{\theta})$ and label y. The generalization performance can be evaluated by the expected risk $\mathbb{E}_{(\boldsymbol{x}, y)} \ell(f(\mathbf{x}, \boldsymbol{\theta}), y)$.

Let $\boldsymbol{\theta} = \mathtt{fine\text{-}tune}(\mathcal{D}^{tr}, \boldsymbol{\theta}_0, h)$ be the model parameters obtained through fine-tuning on the training data $\mathcal{D}^{tr}$ with pre-trained weights $\boldsymbol{\theta}_0$ and a specific hyperparameter configuration h. This hyperparameter configuration can encompass aspects such as the learning rate, weight decay, data augmentation, and random seed, among others [47]. For a set of K hyperparameter configurations $\{h_k\}_{k=1}^K$, let $\boldsymbol{\theta}_k = \mathtt{fine\text{-}tune}(\mathcal{D}^{tr}, \boldsymbol{\theta}_0, h_k)$ denote the model parameters obtained through fine-tuning with the kth configuration h_k. Accordingly, the fine-tuned models $\{\boldsymbol{\theta}_k\}_{k=1}^K$ can be combined together to enhance generalization perfor-

mance. We review representative model soup methods introduced by Wortsman et al. [47]

Uniform-Soup ($\boldsymbol{\theta}_{\mathbf{US}}$) (or SWA [15]) is the most straightforward method to obtain a model soup by uniformly averaging all fine-tuned models:

$$\boldsymbol{\theta}_{\mathrm{US}} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{\theta}_k. \tag{1}$$

This simple averaging approach may not always enhance generalization performance, as the hyperparameters used for fine-tuning are typically randomly searched and can exhibit significant diversity.

Greedy-Soup improves Uniform-Soup by selectively averaging a subset of models. Specifically, it first sorts the fine-tuned models according to their validation accuracies and then sequentially adds models to the soup if the validation performance of the soup is improved. Greedy-Soup empirically outperforms uniform soup and is adopted by practitioners.

Learned-Soup ($\boldsymbol{\theta}_{\mathbf{LS}}$) constructs a model soup by learning coefficients to combine the fine-tuned models. The objective is formulated as follows:

$$\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & \mathcal{L}(\boldsymbol{\theta}_{\mathrm{LS}}; \mathcal{D}^{\mathrm{vl}}) \\
\text{s.t.} \quad & \boldsymbol{\theta}_{\mathrm{LS}} = \alpha_1 \boldsymbol{\theta}_1 + \alpha_2 \boldsymbol{\theta}_2 + \cdots + \alpha_K \boldsymbol{\theta}_K, \\
& \alpha_1 + \alpha_2 + \cdots + \alpha_K = 1, \\
& \alpha_k \in [0,1], \quad k = 1, \ldots, K.
\end{aligned} \tag{2}$$

Solving a constrained optimization problem is challenging. In practice, Wortsman et al. [47] resolve this problem by introducing a parameterization of $\boldsymbol{\alpha}$ using the softmax function, ensuring that each α_k lies in $[0,1]$ and their values sum up to one. The Learned-Soup can be further enhanced by considering the layer-wise structure of deep networks and assigning individual coefficients to each layer. The Learned-Soup is general but needs to load *all* fine-tuned models in the memory and build the computational graph on *all* fine-tuned models for learning $\boldsymbol{\alpha}$, which is infeasible due to memory and computation considerations. Hence, the Learned-Soup is rarely used in practice compared with the Greedy-Soup. In this work, we propose an efficient algorithm to address the memory and computational issues of the Learned-Soup. Different from the Learned-Soup, the proposed algorithm only needs to build the computational graph on the combined model and load a mini-batch of fine-tuned models.

3   Methodology

In this section, we present our memory-efficient learned soup approach. We start by formulating the optimization target as a hyperplane optimization problem (Sec. 3.1), then introduce our efficient coefficient optimization method (Sec. 3.2), and finally employ block coordinate gradient descent to avoid loading all fine-tuned models for learning coefficients (Sec. 3.3).

3.1   Subspace Learning: Hyperplane Optimization Target

For learning the mixing coefficients, the Learned-Soup [47] proposes to learn from a probability simplex using the softmax operation. Specifically, a softmax layer is applied to the learnable variables to output the coefficient vector $\boldsymbol{\alpha}$, which naturally satisfies the convex-hull constraints: $\alpha_k \in [0,1]$ and $\sum_{k=1}^K \alpha_k = 1$. However, recent studies [5, 35] show that interpolation within the convex hull may lead to sub-optimal performance, and extrapolation is more general and can perform better.

Previous computations of the extrapolation weights are normally based on grid search for just two models [5, 35]. It becomes less efficient when merging numerous fine-tuned models as the solution space grows exponentially with the number of models involved. In this work, we develop a novel approach based on subspace learning to seek mixing coefficients that incorporate extrapolation. This relaxes the solution space from a convex hull to a hyperplane spanned by the fine-tuned models $\boldsymbol{\theta}_k$'s. Formally, the objective in Eq. (2) is changed to:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \mathcal{L}(\boldsymbol{\theta}_\star; \mathcal{D}^{\mathrm{vl}}) \\ \text{s.t.} \quad & \boldsymbol{\theta}_\star = \bar{\boldsymbol{\theta}} + \alpha_1(\boldsymbol{\theta}_1 - \bar{\boldsymbol{\theta}}) + \alpha_2(\boldsymbol{\theta}_2 - \bar{\boldsymbol{\theta}}) + \cdots + \alpha_K(\boldsymbol{\theta}_K - \bar{\boldsymbol{\theta}}), \end{aligned} \tag{3}$$

where $\bar{\boldsymbol{\theta}} := \frac{1}{K}\sum_{k=1}^K \boldsymbol{\theta}_k$. One can see that the constraints on $\boldsymbol{\alpha}$ have been removed to allow for extrapolation, and the combined model $\boldsymbol{\theta}_\star$ lies in a hyperplane spanned by $\{\boldsymbol{\theta}_k\}_{k=1}^K$. Note also that Eq. (3) uses $\boldsymbol{\theta}_i - \bar{\boldsymbol{\theta}}$ instead of $\boldsymbol{\theta}_i$, which reduces the correlations among $\boldsymbol{\theta}_i$'s, which also enables better performance (please refer to Appendix D for more details). We refer to this approach as the Hyperplane Learned Soup (HL-Soup) to distinguish it from previous methods focusing on the convex hull.

Note that the mixing coefficients still sum to one, allowing for numerical stability. To see that, $\boldsymbol{\theta}_\star$ in Eq. (3) can be equivalently rewritten as:

$$\boldsymbol{\theta}_\star = \sum_{k=1}^K \left(\frac{1}{K} + \alpha_k - \frac{1}{K}\sum_{k'=1}^K \alpha_{k'} \right) \boldsymbol{\theta}_k, \tag{4}$$

which leads to the identity $\sum_{k=1}^K \left(\frac{1}{K} + \alpha_k - \frac{1}{K}\sum_{k'=1}^K \alpha_{k'} \right) = 1$. By doing so, we can effectively incorporate extrapolation into the weight combination, leading to improved performance.

To further enhance the representation ability of HL-Soup, we introduce a layer-wise mixing scheme called HL-Soup+ as follows:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \mathcal{L}(\boldsymbol{\theta}_\star; \mathcal{D}^{\mathrm{vl}}) \\ \text{s.t.} \quad & \boldsymbol{\theta}_\star^{(l)} = \bar{\boldsymbol{\theta}}^{(l)} + \alpha_1^{(l)}(\boldsymbol{\theta}_1^{(l)} - \bar{\boldsymbol{\theta}}^{(l)}) + \alpha_2^{(l)}(\boldsymbol{\theta}_2^{(l)} - \bar{\boldsymbol{\theta}}^{(l)}) + \cdots + \alpha_K^{(l)}(\boldsymbol{\theta}_K^{(l)} - \bar{\boldsymbol{\theta}}^{(l)}), \\ & l \in \{0, 1, \ldots, L\}. \end{aligned} \tag{5}$$

The use of layer-wise averaging facilitates a more precise manner of model averaging, thereby enhancing the utilization of fine-tuned models and resulting in

better performance. In Sec. 4.3, we will show that in order to achieve similar test performance, the layer-wise approach requires fewer fine-tuned models compared to the greedy soup, which leads to significant computational savings in the fine-tuning stage.

3.2   Efficient Coefficient Optimization

How to efficiently optimize the mixing coefficients $\boldsymbol{\alpha}$ in Eq. (5) is one of the main targets of this work. The classical approach is to construct a computational graph by wrapping all the fine-tuned models and then compute the respective gradient under forward and backward propagation [47]. This method is memory-inefficient due to (i) multiple memory footprints for the internal computational state to build the computational graph; and (ii) all fine-tuned models need to be loaded.

Indeed, the memory issue associated with optimizing the mixing coefficients has been acknowledged as an ongoing challenge [47]. To address this, Wortsman et al. [47] initially propose combining models in CPU rather than GPU, as CPU generally offers larger memory capacities. However, training on the CPU can be substantially slower compared to training on the GPU. Additionally, despite this adjustment, the memory issue is not entirely resolved since the memory capacity of the CPU still remains limited.

In this work, by using the proposed hyperplane optimization, we can conveniently derive the gradient with respect to α_k as follows:

$$\nabla_{\alpha_k}\mathcal{L}(\boldsymbol{\theta}_\star;\mathcal{D}^{\mathrm{vl}}) = \nabla_{\boldsymbol{\theta}_\star}^\top \mathcal{L}(\boldsymbol{\theta}_\star;\mathcal{D}^{\mathrm{vl}})\nabla_{\alpha_k}\boldsymbol{\theta}_\star = \nabla_{\boldsymbol{\theta}_\star}^\top \mathcal{L}(\boldsymbol{\theta}_\star;\mathcal{D}^{\mathrm{vl}})(\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}})\,. \qquad (6)$$

To simplify notations, we consider the non-layer-wise case here. Extension to the layer-wise case is straightforward. The first component $\nabla_{\boldsymbol{\theta}_\star}^\top \mathcal{L}(\boldsymbol{\theta}_\star;\mathcal{D}^{\mathrm{vl}})$ in Eq. (6) is shared across all fine-tuned models, and thus only one computational graph on the model soup $\boldsymbol{\theta}_\star$ is required. The second component, $\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}}$, is specific to the kth fine-tuned model. Hence, $\{\nabla_{\boldsymbol{\alpha}_k}\mathcal{L}(\boldsymbol{\theta}_\star;\mathcal{D}^{\mathrm{vl}})\}_{k=1}^K$ only needs the computational graph on $\boldsymbol{\theta}_\star$, which is affordable for a single GPU. Thus, this addresses the additional memory burden caused by computational graph construction, which typically requires memory that is multiple times the model size (Sec. 4.1). Moreover, the simple inner product operation in Eq. (6) is particularly advantageous for leveraging GPU acceleration. The remaining memory burden is caused by caching all fine-tuned models and will be resolved in the next section.

3.3   Block Coordinate Gradient Descent

To avoid caching all the fine-tuned models, we borrow the classical idea of block coordinate gradient descent (BCGD) [30,37,38,45] for stochastic approximation: We randomly select and update a block of variables by gradient descent at each iteration while keeping the remaining variables fixed. This allows for learning the coefficients without caching all fine-tuned models in memory.

Formally, at iteration t, we sample a mini-batch of b coordinates $\mathcal{K}_t = \{t_1, \ldots, t_b\}$ $\subseteq \{1, \ldots, K\}$ and update $\{\alpha_{k,t} : k \in \mathcal{K}_t\}$ while keeping $\{\alpha_{k,t} : k \notin \mathcal{K}_t\}$ unchanged. Obviously, computation for $\{\nabla_{\alpha_{k,t}} \mathcal{L}(\boldsymbol{\theta}_\star; \mathcal{D}^{\mathrm{vl}}) : k \notin \mathcal{K}_t\}$ at iteration t is not included in BCGD, and only the fine-tuned models corresponding to the chosen coordinates in $\mathcal{K}_t$ are considered. To be specific, let $\boldsymbol{\theta}_{\star,t}$ be the model soup at iteration t. By Eq. (6), the update rule for one gradient descent step of the mixing coefficients can be written as:

$$\alpha_{k,t+1} = \begin{cases} \alpha_{k,t} - \eta \nabla_{\boldsymbol{\theta}_{\star,t}}^\top \mathcal{L}(\boldsymbol{\theta}_{\star,t}; \mathcal{D}^{\mathrm{vl}})(\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}}) & \text{if } k \in \mathcal{K}_t \\ \alpha_{k,t} & \text{if } k \notin \mathcal{K}_t \end{cases} \tag{7}$$

Using the updated coefficients, we update the model soup as:

$$\begin{aligned} \boldsymbol{\theta}_{\star,t+1} &= \bar{\boldsymbol{\theta}} + \sum_{k=1}^K \alpha_{k,t+1}(\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}}) \\ &= \boldsymbol{\theta}_{\star,t} - \underbrace{\sum_{k\in\mathcal{K}_t} \alpha_{k,t}(\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}}) + \sum_{k\in\mathcal{K}_t} \alpha_{k,t+1}(\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}})}_{\text{independent of } \{\boldsymbol{\theta}_{k'} : k' \notin \mathcal{K}_t\}} . \end{aligned} \tag{8}$$

Therefore, the new soup $\boldsymbol{\theta}_{\star,t+1}$ can be constructed from the previous soup $\boldsymbol{\theta}_{\star,t}$ and a weighted combination of the chosen fine-tuned models. In total, we only need to cache $b+1$ models (one model soup and b fine-tuned models), which is much more memory-efficient than the Learned-Soup [47] that requires caching $K + 1$ models. Together with Eq. (7), the mixing coefficients can be learned without the necessity of loading all models, which effectively resolves the memory issue from $O(KD)$ to $O(bD)$, where D is the number of model parameters.

The process of loading/unloading a mini-batch of models into memory at each iteration is time-consuming. To improve efficiency, at each (outer) iteration t, we load the chosen models $\{\boldsymbol{\theta}_k : k \in \mathcal{K}_t\}$ and update the corresponding coefficients $\{\boldsymbol{\alpha}_k : k \in \mathcal{K}_t\}$ for J successive (inner) iterations. The whole procedure of learning layer-wise mixing coefficients, called a Memory-Efficient training algorithm for a Hyperplane Learned Soup (denoted MEHL-Soup+), is shown in Algorithm 1. MEHL-Soup+ is memory-efficient and scalable to large and numerous models.

3.4 Convergence Analysis

In this section, we study the convergence of Algorithm 1. We first make some assumptions that are standard in stochastic optimization [7, 17, 25, 27, 36, 41].

Assumption 1 (Smoothness) $\mathcal{L}(\boldsymbol{\alpha}; \mathcal{D}^{vl})$ *is β-smooth in* $\boldsymbol{\alpha}$, *i.e.,*

$$\|\nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha}; \mathcal{D}^{vl}) - \nabla_{\boldsymbol{\alpha}'} \mathcal{L}(\boldsymbol{\alpha}'; \mathcal{D}^{vl})\| \le \beta \|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|.$$

Assumption 2 (Bounded variance) *There exists $\sigma > 0$ such that*

$$\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}^{vl}} \left\| \nabla_{\boldsymbol{\alpha}} \ell(f(\mathbf{x}; \boldsymbol{\alpha}), y) - \nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha}; \mathcal{D}^{vl}) \right\|^2 \le \sigma^2.$$

Algorithm 1: MEHL-Soup+.

Input: potential soup ingredients $\{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_K\}$, outer iterations T, inter iterations J, model mini-batch size b, #layers L, learning rate η

Output: learned soup $\boldsymbol{\theta}_\star$

1 $\bar{\boldsymbol{\theta}} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{\theta}_k$;

2 initialize $\boldsymbol{\theta}_{\star,0} = \bar{\boldsymbol{\theta}}$, $\boldsymbol{\alpha}_{k,0} = \mathbf{0}$ for $k = 1, \ldots, K$;

3 for $t = 1, \ldots, T$ **do**

4   sample a mini-batch of b coordinates $\mathcal{K}_t = \{t_1, \ldots, t_b\}$;

5   load the fine-tuned models $\{\boldsymbol{\theta}_k\}_{k \in \mathcal{K}_t}$;

6 $\boldsymbol{\theta}_{\text{fix}}^{(l)} = \text{stop\_gradient}\left(\boldsymbol{\theta}_{\star,(t-1)J}^{(l)} - \sum_{k \in \mathcal{K}_t}\alpha_{k,(t-1)J}^{(l)}(\boldsymbol{\theta}_k^{(l)} - \bar{\boldsymbol{\theta}}^{(l)})\right)$ for $l = 0, \ldots, L$;

7   **for** $j = 1, \ldots, J$ **do**

8   compute #iterations $i = (t-1)J + j - 1$;

9 $\boldsymbol{\theta}_{\star,i}^{(l)} = \boldsymbol{\theta}_{\text{fix}}^{(l)} + \sum_{k \in \mathcal{K}_t}\alpha_{k,i}^{(l)}(\boldsymbol{\theta}_k^{(l)} - \bar{\boldsymbol{\theta}}^{(l)})$ for $l = 0, \ldots, L$;

10   sample a mini-batch validation data $\mathcal{B}_i$ from $\mathcal{D}^{\text{vl}}$;

11   calculate gradients $\{\nabla_{\boldsymbol{\alpha}_{k,i}}\mathcal{L}(\boldsymbol{\theta}_{\star,i}; \mathcal{B}_i) : k \in \mathcal{K}_t\}$ by Eq. (6) for $l = 0, \ldots, L$;

12   **for** $k = 1, \ldots, K$ **do**

13   **if** $k \in \mathcal{K}_t$ **then**

14 $\boldsymbol{\alpha}_{k,i+1} = \boldsymbol{\alpha}_{k,i} - \eta\nabla_{\boldsymbol{\alpha}_{k,i}}\mathcal{L}(\boldsymbol{\theta}_{\star,i}; \mathcal{B}_i)$;

15   **else**

16 $\boldsymbol{\alpha}_{k,i+1} = \boldsymbol{\alpha}_{k,i}$;

17 $\boldsymbol{\theta}_{\star,tJ}^{(l)} = \boldsymbol{\theta}_{\text{fix}}^{(l)} + \sum_{k \in \mathcal{K}_t}\alpha_{k,tJ}^{(l)}(\boldsymbol{\theta}_k^{(l)} - \bar{\boldsymbol{\theta}}^{(l)})$ for $l = 0, \ldots, L$;

18 return $\boldsymbol{\theta}_{\star,TJ}$.

Theorem 3. *If the learning rate $\eta \leq \min\{\frac{1}{\beta}, \frac{1}{\sqrt{T}}\}$, Algorithm 1 satisfies*

$$\min_{1 \leq t \leq T} \mathbb{E}\|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ}; \mathcal{D}^{vl})\|^2 \leq \frac{2K\left(\mathbb{E}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,1}; \mathcal{D}^{vl}) - \mathbb{E}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,TJ}; \mathcal{D}^{vl})\right)}{b\sqrt{T}} + \frac{\beta J\sigma^2 K}{b\sqrt{T}},$$

where the expectation is taken over the random mini-batch of samples and models.

The proof can be found in Appendix A. The $O\left(\frac{1}{\sqrt{T}}\right)$ speed matches the convergence rate in [7]. Moreover, we can see that increasing the batch size b of models decreases the upper bound. As a large b intensifies the burden on memory, there is a trade-off between convergence rate and memory-efficiency. When $b = K$, it reduces to the learned soup with extrapolated mixing weights.

4   Experiments

In this section, experiments are performed to demonstrate the efficiency and effectiveness of the proposed methods. We begin by evaluating various model soup methods on the ImageNet dataset. We then conduct experiments specifically targeting a larger model. Finally, we provide a detailed comparison between

Table 1: Comparison of different methods on ImageNet with pre-trained CLIP ViT-B/32. The number of fine-tuned models is 72. We measure the time and memory on a server with one NVIDIA GeForce RTX 4090 GPU and 256 GB RAM.

Method	Testing accuracy (%)	Time per epoch	#epoch	Soup construction time	Peak memory burden
Best individual model	80.38	-	-	-	-
Uniform-Soup [15]	79.97	-	-	-	-
Greedy-Soup [47]	81.03	24s	143[a]	3501s	3GB
Learned-Soup [47]	80.88	1503s	5	7514s	249GB[b]
HL-Soup (**ours**)	81.14	496s	5	2479s	43GB[b]
MEHL-Soup (**ours**)	**81.20**	39s	20[c]	776s	18GB
Learned-Soup+ [47]	81.39	1540s	5	7701s	253GB[b]
HL-Soup+ (**ours**)	81.45	581s	5	2903s	44GB[b]
MEHL-Soup+ (**ours**)	**81.62**	40s	20[c]	808s(↓ 9.5×)	19GB(↓ 13×)
Ensemble[d]	81.19	-	-	-	-

[a] Greedy-Soup involves two evaluation stages: the first stage evaluates all models and sorts them, while the second stage sequentially adds each model to the soup.

[b] We place the fine-tuned models in CPU memory following [47] and report CPU memory usage as it is too large to fit into 24 GB GPU memory.

[c] We use a mini-batch of 18 models and hence the corresponding number of training epochs for MEHL-Soup(+) is 4x longer than those of learned soup and HL-Soup(+).

[d] Ensemble directly utilizes the outputs of all models and does not require soup construction.

greedy and learned soup methods and ablation studies. More experiments on ResNet [10] can be found in Appendix C.

4.1 Experiments on ViT-B/32

Setup. Following [47], we perform experiments on the ImageNet [39] using the pre-trained CLIP ViT-B/32. We use the publicly available fine-tuned models provided by [47]. They are obtained by a random hyperparameter search over the learning rate, weight decay, training epochs, label smoothing, and data augmentation, resulting in a total of 72 fine-tuned models. Training details can be found in Appendix B.

Baselines. The proposed HL-Soup(+) and MEHL-Soup(+) are compared with (i) Best individual model with the highest accuracy on the validation set, (ii) Uniform-Soup [15], which averages all model parameters uniformly, (iii) Greedy-Soup [47], which greedily adds models to the soup to improve validation accuracy, (iv) Learned-Soup [47], which learns coefficients to combine models, and (v) Ensemble, which combines the outputs of all fine-tuned models by aggregating their logit outputs. All methods use the same fine-tuned models and validation set. We utilize the official code provided by [47] for reproducing Greedy-Soup and Learned-Soup.

Results. As can be seen from Tab. 1, MEHL-Soup+ achieves the highest accuracy. Besides, MELH-Soup+ achieves an accuracy gain of 1.24% over the best individual model, demonstrating the effectiveness of learning the model soup.

Comparison with Greedy-Soup. The proposed learned soup approaches outperform Greedy-Soup by an accuracy gain of 0.17% via MEHL-Soup and 0.59% via MEHL-Soup+, respectively. Regarding the soup construction time, MEHL-Soup(+) is 4× faster than Greedy-Soup. Recall that for Greedy-Soup, the number of validation performance evaluations is equal to twice the number of fine-tuned models (one on sorting the fine-tuned models and one for performance evaluation after each candidate model is added to the soup). This can be even larger than the number of training epochs for MEHL-Soup(+), particularly when there are numerous models, e.g., 143 epochs of validation for Learned-Soup(+) and 20 epochs training for MEHL-Soup(+) with 72 models, and thus MEHL-Soup(+) can be faster.

Comparison with Learned-Soup. MEHL-Soup achieves higher accuracy (+0.32%) than Learned-Soup. The source of accuracy improvement may come from the weight extrapolation (i.e., mixing coefficients outsize (0,1)) is more flexible than the Learned-Soup whose coefficients are constrained in $(0,1)$ due to softmax parameterization. This observation also agrees with recent findings [5, 35] that weight extrapolation can boost the performance of combining two models. For the layer-wise scheme, MEHL-Soup+ also performs better than Learned-Soup+. In particular, note that for the non-layerwise scheme, Learned-Soup is worse than Greedy-Soup while the proposed MEHL-Soup still achieves higher accuracy, confirming that learning mixing coefficients from the hyperplane is better than the probability simplex.

Regarding the soup construction time and memory burden, Learned-Soup incurs a significant memory burden, with peak memory reaching as high as 253 GB. This is primarily due to the need to build a computational graph on all fine-tuned models, which severely increases the additional memory overhead. The burden becomes particularly evident when dealing with larger models. Consequently, the process of combining models has to be carried out in CPU memory, significantly slowing down the training speed. Instead, by our subspace training approach and employing a mini-batch coordinate gradient descent strategy, we successfully reduce the memory burden by 13×, allowing efficient training with a single GPU, and then remarkably reduce the corresponding soup construction time by 9.5×.

Comparison with Ensemble. We observe that our learned approach can significantly outperform the model ensemble (e.g., by +0.43% with MEHL-Soup+). Note that ensemble typically requires higher inference costs since they involve aggregating the outputs of all models.

In all, the proposed MEHL-Soup(+) addresses the huge memory requirements of previous methods, enabling scalability and efficient execution on a single GPU. Moreover, MEHL-Soup(+) achieves higher accuracy than Greedy-Soup.

Visualization of Mixing Coefficients. Fig. 1 shows the distributions of mixing

coefficients for all the layers and fine-tuned models learned by MEHL-Soup+ and Learned-Soup+. As can be seen, MEHL-Soup+ learns extrapolated coefficients (i.e., values outside [0, 1]), but Learned-Soup+ enforces the constraint that coefficients must lie in the range $(0, 1)$. The extrapolated coefficients can lead to better performance as demonstrated in Tables 1 and 2. Furthermore, as can be seen, most of the coefficients are close to zero for both methods.
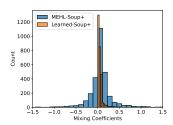

Fig. 1: Distributions of mixing coefficient over all layers and models learned by MEHL-Soup+ and Learned-Soup+ on ImageNet with CLIP ViT-B/32.

4.2   Experiments on Larger ViT-L/14

After resolving the memory issues, the proposed training approach can be used to learn mixing coefficients for combining *larger* models, which is impractical for the previous Learned-Soup method due to the substantial memory requirement. To demonstrate this, we adopt the CLIP ViT-L/14 model[1] [32] and evaluate on three datasets: CIFAR-10, CIFAR-100, and ImageNet. The fine-tuning and training details can be found in Appendix B.

Tab. 2 shows the results. As can be seen, MEHL-Soup+ consistently achieves higher accuracy than Greedy-Soup (+0.12% on CIFAR-10, +0.38% on CIFAR100, and +0.39% on ImageNet). Moreover, when there are 32 fine-tuned models, Learned-Soup requires over 256GB of memory and cannot be run even on a CPU. In contrast, MEHL-Soup+ still remains memory-efficient and can be run on a single GPU.

4.3   Further Comparison between Greedy and Learned Soups

Apart from comparisons of efficiency and performance between different model soup methods, here we delve further into a comprehensive comparison between MEHL-Soup+ and Greedy-Soup from two perspectives: (i) total fine-tuning cost, and (ii) sensitivity to top-performing models, which have not been explored in the previous literature yet are of importance for practical usage.

Test accuracy vs total fine-tuning cost. In previous comparisons, we mainly focus on the model soup stage, where all fine-tuned models are already obtained. However, in many real-world scenarios, these models are fine-tuned with different hyperparameter configurations determined by grid/random search. Typically, the time cost of fine-tuning a single model is much larger than that of the model soup construction. For example, fine-tuning a CLIP ViT-B/32 on ImageNet requires around 4 GPU hours, while model soup training takes less than 1 hour. This is because fine-tuning is performed on the training set, which is

---

[1] ViT-L/14 contains 343M parameters, while ViT-B/32 contains only 87M parameters.

Table 2: Comparison of different methods with pre-trained CLIP ViT-L/14. The number of fine-tuned models is 32 for CIFAR-10/100 and 8 for ImageNet. We measure the time and memory on a server with one NVIDIA GeForce RTX 4090 GPU and 256 GB RAM. "n/a" means the result is not available due to Out-Of-Memory.

	Method	Testing accuracy (%)	Time per epoch	#epoch	Soup construction time	Peak memory burden
CIFAR-10	Best individual model	98.93	-	-	-	-
	Uniform-Soup [15]	99.23	-	-	-	-
	Greedy-Soup [47]	99.15	80s	63	5046s	6GB
	Learned-Soup [47]	n/a	n/a	n/a	n/a	>256GB
	HL-Soup (**ours**)	99.25	1203s	5	6015s	49GB
	MEHL-Soup (**ours**)	99.26	199s	20	3976s	23GB
	Learned-Soup+ [47]	n/a	n/a	n/a	n/a	>256GB
	HL-Soup+ (**ours**)	99.24	1241s	5	6205s	49GB
	MEHL-Soup+ (**ours**)	**99.27**	199s	20	3988s	23GB
	Ensemble	99.14	-	-	-	-
CIFAR-100	Best individual model	92.49	-	-	-	-
	Uniform-Soup [15]	93.05	-	-	-	-
	Greedy-Soup [47]	93.32	81s	63	5084s	6GB
	Learned-Soup [47]	n/a	n/a	n/a	n/a	>256GB
	HL-Soup (**ours**)	93.41	1251s	5	6255s	50GB
	MEHL-Soup (**ours**)	93.52	200s	20	4002s	23GB
	Learned-Soup+ [47]	n/a	n/a	n/a	n/a	>256GB
	HL-Soup+ (**ours**)	93.59	1255s	5	6275s	50GB
	MEHL-Soup+ (**ours**)	**93.70**	205s	20	4100s	23GB
	Ensemble	93.65	-	-	-	-
ImageNet	Best individual model	85.48	-	-	-	-
	Uniform-Soup [15]	85.11	-	-	-	-
	Greedy-Soup [47]	85.64	473s	15	7097s	6GB
	Learned-Soup [47]	85.20	7340s	5	36700s	90GB
	HL-Soup (**ours**)	85.70	950s	5	4748s	23GB
	MEHL-Soup (**ours**)[a]	85.70	950s	5	4748s	23GB
	Learned-Soup+ [47]	85.53	7372s	5	36858s	90GB
	HL-Soup+ (**ours**)	**86.03**	1066s	5	5330s	23GB
	MEHL-Soup+ (**ours**)[a]	**86.03**	1066s	5	5330s(↓ 6.9×)	23GB(↓ 3.9×)
	Ensemble	86.10	-	-	-	-

[a] MEHL-Soup(+) recovers HL-Soup(+) as we use a mini-batch of 8 models, which can fit into a single GPU.

usually much larger than the validation set used in model soup training. Thus, it is necessary to take the cost of fine-tuning stage into consideration.

To investigate this, we gradually increase the number of fine-tuned models and measure the test accuracies achieved by Greedy-Soup and our MEHL-Soup+, as well as the corresponding total fine-tuning costs. Results are shown in

Table 3: Test accuracies (%) of different model soup methods after eliminating different numbers of top-performing models. We sort the fine-tuned CLIP ViT-B/32 models on ImageNet (Sec. 4.1) according to the validation accuracy and compare the performance of Greedy-Soup and MEHL-Soup+ after eliminating 2, 22, and 42 top-performing models.

models eliminated	-	Top-2	Top-22	Top-42
Greedy-Soup	81.03	80.78	80.08	79.70
MEHL-Soup+	81.62(+0.59)	81.58(+0.80)	81.44(+1.36)	81.01(+1.31)

Fig. 2. It is evident that to reach comparable test accuracy, MEHL-Soup+ requires significantly fewer GPU hours than Greedy-Soup. For example, to attain a test accuracy of 81%, Greedy-Soup requires over 200 GPU hours while MEHL-Soup+ takes fewer than 100 GPU hours, a more than 2.5× reduction. This is because MEHL-Soup+ achieves this performance with fewer than 27 fine-tuned models, thanks to its ability to perform layer-wise weighted averaging with much better performance. In contrast, Greedy-Soup requires more than 54 models to achieve similar results. Thus, such efficiency in the fine-tuning cost of MEHL-Soup+ brings further efficiency over Greedy-Soup beyond soup construction.

Sensitivity to top-performing models. To initialize the greedy soup, one selects the fine-tuned model with the best validation performance. The remaining fine-tuned models
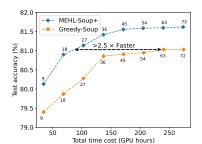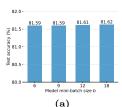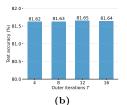

Fig. 2: Test accuracy comparison of Greedy-Soup and MEHL-Soup+ w.r.t. fine-tuning time cost. The experiment is performed on ImageNet with CLIP ViT-B/32. We use different numbers of fine-tuned models (displayed near the points) and measure their corresponding fine-tuning time costs. The model sequence follows the original random search order provided in [47].

are then sequentially tried to be added to the soup following a decreasing order of validation performance. In practice, grid/random search is commonly employed to identify the best fine-tuning hyperparameters [47]. However, obtaining a high-performance model through these search methods is often a challenging task that necessitates numerous trials. Thus, it is important to examine whether such top-performing models are important to the success of greedy soup.

To this end, we replicate the ImageNet experiment in Sec. 4.1 and sort the models in decreasing order of validation accuracy. We then compare the performance of Greedy-Soup and our MEHL-Soup+ after eliminating different numbers of top-performing fine-tuned models. From the results in Tab. 3, we can observe that Greedy-Soup is more sensitive to the top-performing models than MEHL-Soup+. For example, after eliminating the top-2 performance models, Greedy-Soup experiences a significant drop of 0.25% in accuracy, while MEHL-Soup+ shows only a negligible drop. As elimination progresses to 22 and 42 top-
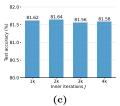
Fig. 3: Sensitivity analysis of the hyperparameters in MEHL-Soup+. The experiments are conducted on ImageNet with CLIP ViT-B/32.

performing models, the advantage of MEHL-Soup+ over Greedy-Soup becomes even more pronounced, resulting in an accuracy gain of 1.36%. Remarkably, even after removing the top-42 performance models, MEHL-Soup+ still achieves an impressive 81% test accuracy. These findings highlight that MEHL-Soup+ exhibits a significantly lower reliance on top-performing models thanks to the better flexibility by learned mixing coefficients. This low reliance holds significant practical value and can help save some effort for hyperparameter search.

4.4   Ablation Studies

Model mini-batch size. We first investigate the effects of model mini-batch size b in MEHL-Soup+. Fig. 3a shows that the accuracy does not vary too much ($<0.1\%$) as the model mini-batch size varies. In practice, one can adjust the model mini-batch size based on the available memory of the training device.

Number of model training iterations. Here we investigate how T, the number of outer iterations, affects accuracy. In Fig. 3b, we observe that as T increases, the performance gain becomes minor. Therefore, in our experiments (Secs. 4.1 to 4.3), we simply use one model training epoch (i.e. $\lceil K/b \rceil$) for efficiency.

Number of inner training iterations. In Secs. 4.1 to 4.3, we use 1K inner iterations (corresponding to 5 epochs over the validation set). We further try 2K, 3K, and 4K iterations in this ablation study. As shown in Fig. 3c, using more inner iterations does not yield a significant performance gain.

5   Conclusion

In this paper, we studied the scaling issue of learning mixing coefficients to build a model soup from numerous fine-tuned models. We proposed a novel approach MEHL-Soup(+) based on efficient hyperplane optimization and block coordinate gradient descent. MEHL-Soup(+) is computation- and memory-efficient and can be run on a single GPU. Moreover, our method allows for extrapolated coefficients, and thus is more expressive than Learned-Soup whose coefficients are constrained in the probability simplex. We also theoretically established the convergence of MEHL-Soup(+). Experimental results on various datasets demonstrate that MEHL-Soup(+) is more efficient and accurate than the Learned-Soup. Furthermore, we hope our strategy could be beneficial to the fine-tuning process by unifying the fine-tuning and model averaging steps under a unified and systematic framework, which would lead to more efficient results.

Acknowledgements

References

1. Cai, R., Zhang, Z., Wang, Z.: Robust weight signatures: Gaining robustness as easy as patching weights? In: International Conference on Machine Learning (ICML) (2023)
2. Camuto, A., Deligiannidis, G., Erdogdu, M.A., Gurbuzbalaban, M., Simsekli, U., Zhu, L.: Fractal structure and generalization properties of stochastic optimization algorithms. In: Advanced in Neural Information Processing Systems (NeurIPS) (2021)
3. Chen, M., Jiang, M., Dou, Q., Wang, Z., Li, X.: FedSoup: Improving generalization and personalization in federated learning via selective model interpolation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCA) (2023)
4. Chronopoulou, A., Peters, M.E., Fraser, A., Dodge, J.: AdapterSoup: Weight averaging to improve generalization of pretrained language models. arXiv preprint arXiv:2302.07027 (2023)
5. Croce, F., Rebuffi, S.A., Shelhamer, E., Gowal, S.: Seasoning model soups for robustness to adversarial and natural distribution shifts. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2023)
6. Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical automated data augmentation with a reduced search space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. pp. 702–703 (2020)
7. Ghadimi, S., Lan, G.: Stochastic first-and zeroth-order methods for nonconvex stochastic programming. SIAM Journal on Optimization (2013)
8. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014)
9. Gressmann, F., Eaton-Rosen, Z., Luschi, C.: Improving neural network training in low dimensional random bases. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
11. Huang, B.: Adversarial Learned Soups: neural network averaging for joint clean and robust performance. Ph.D. thesis, Massachusetts Institute of Technology (2023)
12. Hunter, J.S.: The exponentially weighted moving average. Journal of quality technology (1986)

13. Ilharco, G., Ribeiro, M.T., Wortsman, M., Schmidt, L., Hajishirzi, H., Farhadi, A.: Editing models with task arithmetic. In: International Conference on Learning Representations (ICLR) (2023)
14. Ilharco, G., Wortsman, M., Gadre, S.Y., Song, S., Hajishirzi, H., Kornblith, S., Farhadi, A., Schmidt, L.: Patching open-vocabulary models by interpolating weights. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)
15. Izmailov, P., Wilson, A., Podoprikhin, D., Vetrov, D., Garipov, T.: Averaging weights leads to wider optima and better generalization. In: Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI) (2018)
16. Jiang, W., Kwok, J., Zhang, Y.: Subspace learning for effective meta-learning. In: International Conference on Machine Learning (ICML) (2022)
17. Jiang, W., Yang, H., Zhang, Y., Kwok, J.: An adaptive policy to employ sharpness-aware minimization. In: International Conference on Learning Representations (ICLR) (2023)
18. Kaddour, J.: Stop wasting my time! saving days of imagenet and bert training with latest weight averaging. arXiv preprint arXiv:2209.14981 (2022)
19. Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., Houlsby, N.: Big transfer (bit): General visual representation learning. In: European conference on computer vision (ECCV). Springer (2020)
20. Kornblith, S., Shlens, J., Le, Q.V.: Do better imagenet models transfer better? In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
21. Lei, Y.: Stability and generalization of stochastic optimization with nonconvex and nonsmooth problems. In: The Thirty Sixth Annual Conference on Learning Theory (2023)
22. Li, C., Farkhoor, H., Liu, R., Yosinski, J.: Measuring the intrinsic dimension of objective landscapes. In: International Conference on Learning Representations (ICLR) (2018)
23. Li, T., Huang, Z., Tao, Q., Wu, Y., Huang, X.: Trainable weight averaging: Efficient training by optimizing historical solutions. In: International Conference on Learning Representations (ICLR) (2022)
24. Li, T., Tan, L., Huang, Z., Tao, Q., Liu, Y., Huang, X.: Low dimensional trajectory hypothesis is true: Dnns can be trained in tiny subspaces. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) (2022)
25. Li, T., Tao, Q., Yan, W., Wu, Y., Lei, Z., Fang, K., He, M., Huang, X.: Revisiting random weight perturbation for efficiently improving generalization. Transactions on Machine Learning Research (TMLR) (2024)
26. Li, T., Wu, Y., Chen, S., Fang, K., Huang, X.: Subspace adversarial training. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022)
27. Li, T., Zhou, P., He, Z., Cheng, X., Huang, X.: Friendly sharpness-aware minimization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2024)
28. Liu, T.Y., Soatto, S.: Tangent model composition for ensembling and continual fine-tuning. In: IEEE/CVF International Conference on Computer Vision (CVPR) (2023)
29. Melis, G.: Two-tailed averaging: Anytime adaptive once-in-a-while optimal iterate averaging for stochastic optimization. arXiv preprint arXiv:2209.12581 (2022)
30. Nesterov, Y.: Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM Journal on Optimization (2012)

31. Ortiz-Jimenez, G., Favero, A., Frossard, P.: Task arithmetic in the tangent space: Improved editing of pre-trained models. In: Advanced in Neural Information Processing Systems (NeurIPS) (2023)
32. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International Conference on Machine Learning (ICML) (2021)
33. Rame, A., Kirchmeyer, M., Rahier, T., Rakotomamonjy, A., Gallinari, P., Cord, M.: Diverse weight averaging for out-of-distribution generalization. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)
34. Ramé, A., Vieillard, N., Hussenot, L., Dadashi, R., Cideron, G., Bachem, O., Ferret, J.: Warm: On the benefits of weight averaged reward models. arXiv preprint arXiv:2401.12187 (2024)
35. Rebuffi, S.A., Croce, F., Gowal, S.: Revisiting adapters with adversarial training. In: International Conference on Learning Representations (ICLR) (2023)
36. Reddi, S.J., Hefny, A., Sra, S., Poczos, B., Smola, A.: Stochastic variance reduction for nonconvex optimization. In: International Conference on Machine Learning (ICML) (2016)
37. Richtárik, P., Takáč, M.: Distributed coordinate descent method for learning with big data. Journal of Machine Learning Research (2016)
38. Richtárik, P., Takáč, M.: Parallel coordinate descent methods for big data optimization. Mathematical Programming (2016)
39. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ICMLNet large scale visual recognition challenge. International Journal of Computer Vision (IJCV) (2015)
40. Sanyal, S., Neerkaje, A.T., Kaddour, J., Kumar, A., et al.: Early weight averaging meets high learning rates for llm pre-training. In: Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023) (2023)
41. Si, D., Yun, C.: Practical sharpness-aware minimization cannot converge all the way to optima. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
42. Smith, S., Elsen, E., De, S.: On the generalization benefit of noise in stochastic gradient descent. In: International Conference on Machine Learning (ICML) (2020)
43. Suzuki, K., Matsuzawa, T.: Model soups for various training and validation data. AI (2022)
44. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
45. Tseng, P., Yun, S.: Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. Journal of optimization theory and applications (2009)
46. Wortsman, M., Horton, M., Guestrin, C., Farhadi, A., Rastegari, M.: Learning neural network subspaces. In: International Conference on Machine Learning (ICML) (2021)
47. Wortsman, M., Ilharco, G., Gadre, S.Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A.S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al.: Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In: International Conference on Machine Learning (ICML) (2022)

48. Wright, S.J.: Coordinate descent algorithms. Mathematical programming (2015)
49. Yin, L., Liu, S., Fang, M., Huang, T., Menkovski, V., Pechenizkiy, M.: Lottery pools: Winning more by interpolating tickets without increasing training or inference cost. In: Proceedings of the AAAI Conference on Artificial Intelligence (2023)
50. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in Neural Information Processing Systems (NeurIPS) (2014)
51. Yu, L., Jiang, W., Shi, H., YU, J., Liu, Z., Zhang, Y., Kwok, J., Li, Z., Weller, A., Liu, W.: Metamath: Bootstrap your own mathematical questions for large language models. In: International Conference on Learning Representations (ICLR) (2024)
52. Zhou, Z.H.: Ensemble methods: foundations and algorithms. CRC press (2012)
53. Zimmer, M., Spiegel, C., Pokutta, S.: Sparse model soups: A recipe for improved pruning via model averaging. In: International Conference on Learning Representations (ICLR) (2024)

A   Proofs

Proof (Proof of Theorem 3). At each outer iteration t, let $\mathcal{K}_t = \{t_1, \ldots, t_b\}$ be the indices of models being chosen. For $j = 1, \ldots, J$, it follows from Taylor approximation that $(i = (t-1)J + j)$,

$$
\begin{aligned}
&\mathcal{L}(\boldsymbol{\alpha}_{\cdot,i+1}; \mathcal{D}^{\text{vl}}) \\
&= \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{D}^{\text{vl}}) + \nabla_{\boldsymbol{\alpha}_{\cdot,i}}^{\top} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{D}^{\text{vl}})(\boldsymbol{\alpha}_{\cdot,i+1} - \boldsymbol{\alpha}_{\cdot,i}) \\
&\quad + \frac{1}{2}(\boldsymbol{\alpha}_{\cdot,i+1} - \boldsymbol{\alpha}_{\cdot,i})^{\top} \nabla_{\boldsymbol{\alpha}_{\cdot,i}}^{2} \mathcal{L}(\boldsymbol{\xi}_i; \mathcal{D}^{\text{vl}})(\boldsymbol{\alpha}_{\cdot,i+1} - \boldsymbol{\alpha}_{\cdot,i}) && (9) \\
&\leq \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{D}^{\text{vl}}) - \eta \nabla_{\boldsymbol{\alpha}_{\cdot,i}}^{\top} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{D}^{\text{vl}}) \mathbf{U}_i \nabla \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{B}_i) \\
&\quad + \frac{\eta^2 \beta}{2} \nabla^{\top} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{B}_i) \mathbf{U}_i^2 \nabla \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{B}_i) && (10) \\
&= \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{D}^{\text{vl}}) - \eta \sum_{k=1}^{K} \mathbb{I}(k \in \mathcal{K}_t) \nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{D}^{\text{vl}}) \nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{B}_i) \\
&\quad + \frac{\eta^2 \beta}{2} \sum_{k=1}^{K} \mathbb{I}(k \in \mathcal{K}_t) \left(\nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{B}_i) \right)^2 && (11)
\end{aligned}
$$

where $\boldsymbol{\xi}_i \in [\boldsymbol{\alpha}_{\cdot,i}, \boldsymbol{\alpha}_{\cdot,i+1}]$ in Eq. (9), $\mathbf{U}_i = \text{diag}([\mathbb{I}(1 \in \mathcal{K}_t), \mathbb{I}(2 \in \mathcal{K}_t), \ldots, \mathbb{I}(K \in \mathcal{K}_t)]) \in \mathbb{R}^{K \times K}$ (the kth diagonal entry indicates whether the kth model is chosen at iteration $i \in \{t(J-1), \ldots, tJ\}$), Eq. (10) follows from the smoothness assumption and the update rule of $\boldsymbol{\alpha}_{\cdot,i+1}$. Taking expectation w.r.t. $\mathcal{B}_i$ on both sides of Eq. (11), we obtain

$$
\begin{aligned}
&\mathbb{E}_{\mathcal{B}_i} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i+1}; \mathcal{D}^{\text{vl}}) \\
&\leq \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{D}^{\text{vl}}) - \eta \sum_{k=1}^{K} \mathbb{I}(k \in \mathcal{K}_t) \left(\nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{D}^{\text{vl}}) \right)^2 \\
&\quad + \frac{\eta^2 \beta}{2} \sum_{k=1}^{K} \mathbb{I}(k \in \mathcal{K}_t) \left(\left(\nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{D}^{\text{vl}}) \right)^2 + \text{Var}_{\mathcal{B}_i}(\nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{B}_i)) \right) && (12) \\
&\leq \mathcal{L}(\boldsymbol{\alpha}_{\cdot,i}; \mathcal{D}^{\text{vl}}) - \frac{\eta}{2} \sum_{k=1}^{K} \mathbb{I}(k \in \mathcal{K}_t) \left(\nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{D}^{\text{vl}}) \right)^2 \\
&\quad + \frac{\eta^2 \beta}{2} \sum_{k=1}^{K} \mathbb{I}(k \in \mathcal{K}_t) \text{Var}_{\mathcal{B}_i}(\nabla_{\alpha_{k,i}} \mathcal{L}(\alpha_{k,i}; \mathcal{B}_i)) && (13)
\end{aligned}
$$

where $\text{Var}(x)$ is the variance of x, Eq. (12) follows from the identity $\mathbb{E}x^2 = (\mathbb{E}x)^2 + \text{Var}(x)$, and Eq. (13) follows from the assumption $\eta \leq \frac{1}{\beta}$ (thus, $\frac{\eta^2 \beta}{2} \leq \frac{\eta}{2}$).

Taking expectation w.r.t. $\boldsymbol{\alpha}_{\cdot,i}$ and $\boldsymbol{\alpha}_{\cdot,i+1}$ on both sides of Eq. (13), we have

$$\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,i+1}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,i+1};\mathcal{D}^{\mathrm{vl}}) \leq \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,i}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,i};\mathcal{D}^{\mathrm{vl}}) - \frac{\eta}{2}\sum_{k=1}^{K}\mathbb{I}(k \in \mathcal{K}_t)\mathbb{E}_{\alpha_{k,i}}\big(\nabla_{\alpha_{k,i}}\mathcal{L}(\alpha_{k,i};\mathcal{D}^{\mathrm{vl}})\big)^2$$

$$+ \frac{\eta^2\beta}{2}\sum_{k=1}^{K}\mathbb{I}(k \in \mathcal{K}_t)\mathbb{E}_{\alpha_{k,i}}\mathsf{Var}_{\mathcal{B}_i}\big(\nabla_{\alpha_{k,i}}\mathcal{L}(\alpha_{k,i};\mathcal{B}_i)\big) \qquad (14)$$

Summing over $j = 1 \to J$ and $t = 1 \to T$, let $i = (t-1)J + j$, we obtain

$$\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,TJ}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,TJ};\mathcal{D}^{\mathrm{vl}})$$

$$\leq \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,1}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,1};\mathcal{D}^{\mathrm{vl}}) - \frac{\eta}{2}\sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{k=1}^{K}\mathbb{I}(k \in \mathcal{K}_t)\mathbb{E}_{\alpha_{k,i}}\big(\nabla_{\alpha_{k,i}}\mathcal{L}(\alpha_{k,i};\mathcal{D}^{\mathrm{vl}})\big)^2$$

$$+ \frac{\eta^2\beta}{2}\sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{k=1}^{K}\mathbb{I}(k \in \mathcal{K}_t)\mathbb{E}_{\alpha_{k,i}}\mathsf{Var}_{\mathcal{B}_i}\big(\nabla_{\alpha_{k,i}}\mathcal{L}(\alpha_{k,i};\mathcal{B}_i)\big)$$

$$= \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,1}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,1};\mathcal{D}^{\mathrm{vl}}) - \frac{\eta}{2}\sum_{t=1}^{T}\sum_{k=1}^{K}\mathbb{I}(k \in \mathcal{K}_t)\left(\sum_{j=1}^{J}\mathbb{E}_{\alpha_{k,i}}\big(\nabla_{\alpha_{k,i}}\mathcal{L}(\alpha_{k,i};\mathcal{D}^{\mathrm{vl}})\big)^2\right)$$

$$+ \frac{\eta^2\beta\sigma^2 TJ}{2} \qquad (15)$$

$$\leq \mathbb{E}_{\alpha_{k,1}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,1};\mathcal{D}^{\mathrm{vl}}) - \frac{\eta}{2}\sum_{t=1}^{T}\sum_{k=1}^{K}\mathbb{I}(k \in \mathcal{K}_t)\mathbb{E}_{\alpha_{k,tJ}}\big(\nabla_{\alpha_{k,tJ}}\mathcal{L}(\alpha_{k,tJ};\mathcal{D}^{\mathrm{vl}})\big)^2$$

$$+ \frac{\eta^2\beta\sigma^2 TJ}{2} \qquad (16)$$

where Eq. (15) follows from $\sum_{k=1}^{K}\mathbb{I}(k \in \mathcal{K}_t)\mathbb{E}_{\alpha_{k,i}}\mathsf{Var}_{\mathcal{B}_i}(\nabla_{\alpha_{k,i}}\mathcal{L}(\alpha_{k,i};\mathcal{B}_i)) \leq \sigma^2$, Eq. (16) follows from

$$\sum_{j=1}^{J}\mathbb{E}_{\alpha_{k,i}}\big(\nabla_{\alpha_{k,i}}\mathcal{L}(\alpha_{k,i};\mathcal{D}^{\mathrm{vl}})\big)^2 \geq \mathbb{E}_{\alpha_{k,tJ}}\big(\nabla_{\alpha_{k,tJ}}\mathcal{L}(\alpha_{k,tJ};\mathcal{D}^{\mathrm{vl}})\big)^2.$$

Taking expectation w.r.t. $\mathbb{I}(k \in \mathcal{K}_t)$, we have

$$\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,TJ}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,TJ};\mathcal{D}^{\mathrm{vl}})$$

$$\leq \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,1}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,1};\mathcal{D}^{\mathrm{vl}}) - \frac{\eta b}{2K}\sum_{t=1}^{T}\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,tJ}}\|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ};\mathcal{D}^{\mathrm{vl}})\|^2 + \frac{\eta^2\beta\sigma^2 TJ}{2} \quad (17)$$

$$\leq \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,1}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,1};\mathcal{D}^{\mathrm{vl}}) - \frac{\eta T b}{2K}\min_{1\leq t\leq T}\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,tJ}}\|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ};\mathcal{D}^{\mathrm{vl}})\|^2 + \frac{\eta^2\beta\sigma^2 TJ}{2} \quad (18)$$

where Eq. (17) follows from $\mathbb{E}\mathbb{I}(k \in \mathcal{K}_t) = \frac{b}{K}$ and Eq. (18) follows from

$$\min_{1\leq t\leq T}\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,tJ}}\|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ};\mathcal{D}^{\mathrm{vl}})\|^2 \leq \sum_{t=1}^{T}\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,tJ}}\|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}}\mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ};\mathcal{D}^{\mathrm{vl}})\|^2.$$

Rearranging Eq. (18), it follows that

$$\frac{\eta T b}{2K} \min_{1 \leq t \leq T} \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,tJ}} \|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ}; \mathcal{D}^{\mathrm{vl}})\|^2$$

$$\leq \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,1}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,1}; \mathcal{D}^{\mathrm{vl}}) - \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,TJ}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,TJ}; \mathcal{D}^{\mathrm{vl}}) + \frac{\eta^2 \beta \sigma^2 T J}{2} \qquad (19)$$

Let $\eta = \min\{\frac{1}{\sqrt{T}}, \frac{1}{\beta}\}$, from Eq. (19), we have,

$$\frac{b\sqrt{T}}{2K} \min_{1 \leq t \leq T} \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,tJ}} \|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ}; \mathcal{D}^{\mathrm{vl}})\|^2$$

$$\leq \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,1}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,1}; \mathcal{D}^{\mathrm{vl}}) - \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,TJ}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,TJ}; \mathcal{D}^{\mathrm{vl}}) + \frac{\beta J \sigma^2}{2} \qquad (20)$$

Diving both sides of the above inequality by $\frac{b\sqrt{T}}{2K}$, we obtain

$$\min_{1 \leq t \leq T} \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,tJ}} \|\nabla_{\boldsymbol{\alpha}_{\cdot,tJ}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,tJ}; \mathcal{D}^{\mathrm{vl}})\|^2$$

$$\leq \frac{2K \left(\mathbb{E}_{\boldsymbol{\alpha}_{\cdot,1}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,1}; \mathcal{D}^{\mathrm{vl}}) - \mathbb{E}_{\boldsymbol{\alpha}_{\cdot,TJ}} \mathcal{L}(\boldsymbol{\alpha}_{\cdot,TJ}; \mathcal{D}^{\mathrm{vl}}) \right)}{b\sqrt{T}} + \frac{\beta J \sigma^2 K}{b\sqrt{T}}, \qquad (21)$$

and we finish the proof.

B   Training Details

Model Fine-tuning. The CLIP ViT-B/32 models fine-tuned on ImageNet are publicly available and can be found in `https://github.com/mlfoundations/model-soups`. There are 72 models fine-tuned with a random hyperparameter search over the learning rate, weight decay, training epochs, label smoothing, and data augmentation. The CLIP ViT-L/14 models are fine-tuned with random search over learning rates of $\{1 \times 10^{-6}, 3 \times 10^{-6}, 5 \times 10^{-6}, 1 \times 10^{-5}, 2 \times 10^{-5}, 3 \times 10^{-5}\}$ and weight decays of $\{0, 0.001, 0.003, 0.005, 0.01, 0.03\}$. We fine-tune 32 models for CIFAR-10/100 and 8 models for ImageNet, as the latter requires substantially higher computational costs for fine-tuning. We use a batch size of 128 and set the training epoch to 10. All training images are resized to 224×224 with standard CLIP-style image preprocessing [32], while for ImageNet, we additionally apply RandAugmentation [6] following [47]. We use a random split of 10% training data as the held-out validation set for CIFAR-10/100 and 2% for ImageNet. We will release the fine-tuned model checkpoints.

Learned Soup Training. We train HL-Soup(+) and MEHL-Soup(+) for 1 model training epochs, i.e., loading K models into memory for once, and use a model batch size of $b = 18$ for CLIP ViT-B/32 and $b = 8$ for CLIP ViT-L/14. This corresponds to outer iterations of $T = \lceil K/b \rceil$. We use a batch size of 128 for training. To accommodate ViT-L/4 and its memory requirements within a 24GB GPU, we employ 8 times of gradient accumulation. For HL-Soup(+), MEHL-Soup(+), and Learn-Soup(+), we use 1K inner iterations and adopt

the AdamW optimizer with a cosine learning rate schedule. The learning rate is searched over $\{0.005, 0.01, 0.05\}$ and we choose 0.01 for HL-Soup($+$) and MEHL-Soup($+$), and 0.05 for Learn-Soup($+$). Weight decay is searched over $\{0, 0.001, 0.01, 0.05, 0.1, 0.5\}$ and we use 0.1 for HL-Soup($+$) and MEHL-Soup($+$), and 0 for Learn-Soup($+$) for optimal.

C   More Results

To further evaluate the performance of our approach on different architectures, we conduct an experiment on CIFAR-100 with 32 models fine-tuned from ResNet-101 (pre-trained on ImageNet). The setting is the same as in Tab. 2. The table below shows that MEHL-Soup+ is more efficient and effective than Learned-Soup+.

Table A4: Comparison of different methods with pre-trained ResNet-101 on CIFAR-100. The number of fine-tuned models is 32. We measure the time and memory on a server with one NVIDIA GeForce RTX 4090 GPU and 256 GB RAM.

Method	Testing accuracy (%)	Time per epoch	#epoch	Soup construction time	Peak memory burden
Best individual	87.10	-	-	-	-
Uniform-Soup [15]	86.47	-	-	-	-
Greedy-Soup [47]	88.62	5s	63	328s	4GB
Learned-Soup [47]	88.50	913s	5	4566s	68GB
HL-Soup (**ours**)	88.82	47s	5	238s	17GB
MEHL-Soup (**ours**)	89.03	10s	20	207s	13GB
Learned-Soup+ [47]	88.83	915s	5	4576s	68GB
HL-Soup+ (**ours**)	89.12	48s	5	244s	17GB
MEHL-Soup+ (**ours**)	**89.32**	10s	20	209s ($\downarrow 21.9\times$)	13GB ($\downarrow 5.2\times$)
Ensemble	89.61	-	-	-	-

D   Ablation on Weight Decentralization

Learning with $\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}}$, i.e., weight decentralization, as in Eq. (3) can be more effective and produce better performance. We conduct an ablation (with the same setting as in Tab. 1) to compare MEHL-Soup+ with simply using $\boldsymbol{\theta}'_\star = \sum_{k=1}^K \alpha_k \boldsymbol{\theta}_k$ (without weight decentralization). The test accuracy drops significantly from 81.62 to 81.02. This is perhaps because $\{\boldsymbol{\theta}_k\}_{k=1}^K$ are in the same basin and decomposing them as $\{\boldsymbol{\theta}_k - \bar{\boldsymbol{\theta}}\}_{k=1}^K$ can reduce correlation (the averaged cosine similarity between different weight vectors is decreased from 0.99 to 0.19) for better optimization.