Adapter Merging with Centroid Prototype Mapping for Scalable Class-Incremental Learning

Takuma Fukuda
Chiba University
takuma.fukuda@chiba-u.jp

Hiroshi Kera
Chiba University
Zuse Institute Berlin
kera@chiba-u.jp

Kazuhiko Kawamoto
Chiba University
kawa@faculty.chiba-u.jp

Abstract

We propose Adapter Merging with Centroid Prototype Mapping (ACMap), an exemplar-free framework for class-incremental learning (CIL) that addresses both catastrophic forgetting and scalability. While existing methods involve a trade-off between inference time and accuracy, ACMap consolidates task-specific adapters into a single adapter, thus achieving constant inference time across tasks without sacrificing accuracy. The framework employs adapter merging to build a shared subspace that aligns task representations and mitigates forgetting, while centroid prototype mapping maintains high accuracy by consistently adapting representations within the shared subspace. To further improve scalability, an early stopping strategy limits adapter merging as tasks increase. Extensive experiments on five benchmark datasets demonstrate that ACMap matches state-of-the-art accuracy while maintaining inference time comparable to the fastest existing methods. The code is available at https://github.com/tf63/ACMap.

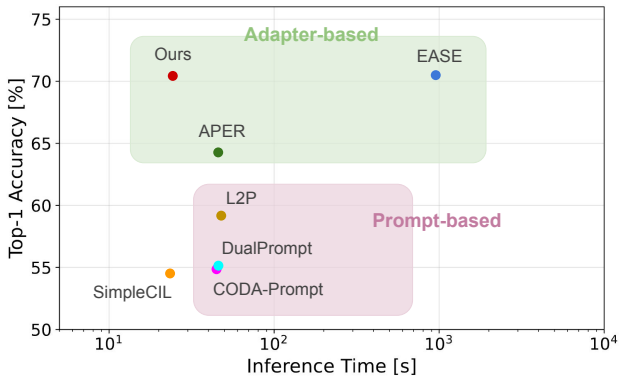


Figure 1. Comparison of the final top-1 accuracy and inference time for task 40 of ImageNet-R in class-incremental learning. The comparison includes L2P [50], DualPrompt [49], CODA-Prompt [41], SimpleCIL [61], APER [61], EASE [59], and our method. All methods use the same backbone (ViT-B/16). Our method performs well in terms of both inference time and accuracy by consolidating task-specific adapters into a single adapter.

1. Introduction

In real-world applications, data often arrives sequentially, which requires continual learning [44] to adapt to evolving data distributions. Class-incremental learning (CIL) [36] is a branch of continual learning designed for scenarios where tasks with new classes appear sequentially. A primary challenge in CIL is *catastrophic forgetting* [10]—the difficulty of learning new tasks while preserving knowledge from previous ones. Traditional CIL methods mitigate catastrophic forgetting by retaining representative data (exemplars) from previous tasks [3, 29, 38] or by dynamically adjusting network structures [48, 54, 57]. However, privacy concerns [39] often limit the use of exemplars. This limitation highlights the need for exemplar-free methods in practical applications.

In contrast to traditional approaches, recent CIL methods [58] using pre-trained models have attracted attention. These methods leverage the strong generalization of pre-trained models to mitigate catastrophic forgetting. Typically, these methods incorporate parameter-efficient modules for task-specific training on each CIL task, such as prompts [21] or adapters [33]. Although effective for domain adaptation, task-specific classifiers often hinder scalability at inference. Figure 1 compares the final top-1 accuracy and inference time on task 40 of ImageNet-R for these methods. The results demonstrate a trade-off between accuracy and inference time. This trade-off highlights the challenge of achieving both high accuracy and scalability.

To address the dual challenges of catastrophic forgetting and scalability in CIL, we propose **A**dapter Merging with **C**entroid Prototype **Map**ping (**ACMap**), a framework that consolidates task-specific adapters into a single adapter. ACMap enables scalability by maintaining con-

stant inference time, without sacrificing accuracy. Furthermore, ACMap's exemplar-free design addresses privacy concerns commonly associated with traditional exemplar-based methods.

ACMap consists of two components: adapter merging and centroid prototype mapping. Adapter merging incrementally combines task-specific adapters into a shared subspace by averaging their weights. This shared subspace mitigates catastrophic forgetting by aligning tasks within the parameter space. While simple weight averaging alone may not ensure optimal performance, ACMap enhances alignment by initializing each adapter from a common starting weight. This initialization promotes consistent training paths across tasks, helping form a low-loss basin in the parameter space. Centroid prototype mapping further supports ACMap's effectiveness by preserving previously learned representations through consistent adaptation across the shared subspace. To further improve scalability, early stopping limits adapter merging, thereby reducing training costs. Evaluations on five benchmark datasets demonstrate that ACMap simultaneously achieves state-of-the-art accuracy and efficient inference. Specifically, on task 40 of ImageNet-R, ACMap improves final accuracy by more than 16% compared to the fastest existing method with similar inference speed, while achieving a 39-fold speedup over the state-of-the-art method with comparable accuracy (see Figure 1).

In summary, our contributions are as follows:

1. We propose ACMap, a continual learning framework that consolidates task-specific adapters into a single shared subspace without storing previous data samples, effectively mitigating catastrophic forgetting.
2. To preserve previously learned representations, we incorporate centroid prototype mapping, ensuring consistency across tasks through adaptive subspace alignment.
3. Extensive experiments on five benchmark datasets demonstrate that ACMap achieves performance comparable to the state-of-the-art in both speed and accuracy, validating its effectiveness and scalability for real-world applications.

2. Related Work

This section discusses traditional class-incremental learning methods and recent approaches with pre-trained models.

Class-Incremental Learning (CIL): Class-incremental learning (CIL) [60] is a learning paradigm in which a model incrementally learns new class information without forgetting previously learned classes. A major challenge in CIL is *catastrophic forgetting* [10, 22], where learning new classes overwrites previously acquired knowledge. This often results in significant performance degradation on earlier tasks. Prior work addresses catastrophic forgetting via three main approaches [60]. The first approach [3, 4, 28, 29, 32, 38, 45] selects and retains representative data (*exemplars*) from previously learned classes. The second approach [2, 9, 34, 48, 54, 57] dynamically modifies the model architecture to accommodate new class information. The third approach [6, 8, 18, 26, 36, 40] leverages knowledge distillation [16] to transfer knowledge from previously learned classes. However, even with these methods, catastrophic forgetting still leads to significant performance degradation. Additionally, many of these approaches rely on the use of exemplars, which can pose challenges related to privacy or storage constraints [60]. Therefore, unresolved issues remain for real-world applications.

CIL with Pre-Trained Models: Recently, there has been growing interest in utilizing large pre-trained models for CIL [12, 58]. In these studies, parameter-efficient modules [13] are commonly employed to learn new tasks while preserving the strong generalization capabilities of the pre-trained model. Two primary approaches have emerged: (i) using learnable parameters (*prompts*) [21] concatenated to input vectors in pre-trained models, and (ii) incorporating adapter modules (e.g., LoRA [17]) into pre-trained models.
(i) Prompt-based Approaches: The first approach focuses on learning prompts for each task using visual prompt tuning (VPT) [21]. VPT introduces learnable prompts into the input of a frozen pre-trained model, tuning only the prompts. L2P [50] is one such method that employs VPT, retrieving instance-specific prompts from a learned prompt pool through key-query matching. DualPrompt [49] utilizes both task-agnostic and task-specific prompts, while CODA-Prompt [41] retrieves prompts from a prompt pool using an attention-based weighted combination. These exemplar-free methods outperform those without pre-trained models.
(ii) Adapter-based Approaches: The second approach uses parameter-efficient adapter modules, such as LoRA. SimpleCIL [61] is a foundational method that constructs a cosine classifier [11] from the average of class-specific feature vectors (*prototype*) [42] extracted from a pre-trained model using validation data. Despite its simplicity, SimpleCIL performs comparably to VPT-based methods. APER [61] builds on SimpleCIL by using a pre-trained model with an adapter trained on the first task. EASE [59] achieves state-of-the-art performance by using task-specific adapters to extract prototypes for each task and constructing a cosine classifier from the concatenated prototypes. In an exemplar-free setting, where prototypes from previous classes cannot be extracted, EASE compensates using cosine similarity-based mapping. While APER has limited domain adaptation capabilities due to training an adapter only for the first task, EASE shows high adaptability by training adapters for all tasks. However, EASE incurs higher inference costs as the number of tasks grows, since feature vectors are extracted using each adapter individu-

ally. To address this scalability issue, our method merges multiple adapters into a single one.

CIL with model merging: Several recent methods have explored model merging in CIL. iTAML [35] and TKIL [52] adopt merging strategies, but follow exemplar-based CIL, unlike ACMap's exemplar-free design. The assumptions and challenges in exemplar-based and exemplar-free settings differ considerably. In particular, exemplar-free methods like ACMap must overcome the absence of memory replay. RAPF [19] and ACMap both rely on adapter merging but take different approaches. While RAPF employs a computationally intensive SVD-based merging method, ACMap uses simple averaging, making it more lightweight. Moreover, RAPF leverages CLIP's text embeddings to enhance feature alignment, whereas ACMap is a vision-only method, better suited for scenarios without multi-modal supervision. Although both rely on merging, effective feature alignment remains crucial to strong performance.

3. Preliminaries

This section introduces the problem formulation of CIL and provides the background on CIL approaches with pre-trained models.

3.1. Problem Setting

Class-Incremental Learning: Class-incremental learning is a learning paradigm where a model is required to sequentially learn a series of T task datasets, $\mathcal{D}_1, \ldots, \mathcal{D}_T$, while retaining knowledge of previously learned tasks. Each dataset $\mathcal{D}_t = \{(\boldsymbol{x}_t, y_t)\}$ consists of input data $\boldsymbol{x}_t \in \mathcal{X}$ and corresponding class labels $y_t \in \mathcal{Y}_t$.[1] For any two distinct tasks t and t', the class sets are disjoint, i.e., $\mathcal{Y}_t \cap \mathcal{Y}_{t'} = \varnothing$. The objective in the t-th task is to learn a model $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}_t$, parameterized by $\boldsymbol{\theta}$, that accurately maps inputs to their class labels. During testing on t-th task, the model is evaluated on the cumulative test dataset $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_t$, where $\mathcal{T}_i$ is the test dataset for the i-th task.

Exemplar and Exemplar-Free CIL: Many traditional CIL approaches retain a subset of representative data from previous tasks, known as an exemplar set. This set for the t-th task is denoted $\mathcal{E}_t = \{(\boldsymbol{x}^{\mathrm{e}}, y^{\mathrm{e}})\}$. In examplar-based CIL, the training dataset for $f_{\boldsymbol{\theta}}$ includes $\mathcal{D}_t$ and exemplars from previous tasks, combined as $\mathcal{D}_t \cup \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_t$. However, privacy concerns and other constraints often limit the use of exemplars. In exemplar-free settings, $f_{\boldsymbol{\theta}}$ is trained exclusively on the current task dataset $\mathcal{D}_t$. We evaluate our approach under exemplar-free conditions for broader applicability in privacy-sensitive scenarios.

[1] We drop the index i from $(\boldsymbol{x}_{t,i}, y_{t,i})$ for notational simplicity.

3.2. Pre-Trained Models for CIL

Following previous studies [59, 61], we utilize a pre-trained Vision Transformer (ViT) [7, 46] to initialize f. The model is decomposed into a linear classifier $\boldsymbol{W} \in \mathbb{R}^{d \times |\mathcal{Y}_t|}$ and a feature embedding function $\phi : \mathbb{R}^D \to \mathbb{R}^d$, where D is the dimension of the input vector and d is the embedding dimension. The function ϕ denotes the final [CLS] token embedding in ViT, which represents the global image representation. For an input $\boldsymbol{x} \in \mathbb{R}^D$, the model output is given by $f(\boldsymbol{x}) = \boldsymbol{W}^T \phi(\boldsymbol{x})$.

Adapter-based CIL: Trainable parameter-efficient adapter modules are often employed when applying a pre-trained model to a task. The adapter has a bottleneck structure, consisting of a down-projection layer $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d \times r}$ and an up-projection layer $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{r \times d}$, where r is the bottleneck dimension and satisfies $r \ll d$. To introduce non-linearity, a ReLU layer is positioned between the projection layers. The adapter is added to the MLP layer via a residual connection. Given the input of the MLP layer as $\boldsymbol{x}_{\text{in}} \in \mathbb{R}^{d \times d}$, the modified output $\boldsymbol{x}_{\text{out}} \in \mathbb{R}^{d \times d}$ with the adapter becomes:

$$\boldsymbol{x}_{\text{out}} = \text{MLP}(\boldsymbol{x}_{\text{in}}) + \text{ReLU}(\boldsymbol{x}_{\text{in}} \boldsymbol{W}_{\text{down}}) \boldsymbol{W}_{\text{up}}. \tag{1}$$

The adapter is inserted across each N_{blocks} transformer block. We refer to these N_{blocks} adapters collectively as "the adapter", denoted as $\mathcal{A}$. In adapter-based CIL, a task-specific subspace is formed by training an adapter $\mathcal{A}_t$ for each task t.

Prototypical Classification in CIL: After training the adapter on the t-th task, a prototypical classifier is constructed using the t-th validation dataset $\mathcal{V}_t$. Specifically, we calculate the prototype $\boldsymbol{p}_{t,c} \in \mathbb{R}^d$, which is the mean of the feature vectors for each class $c \in \mathcal{Y}_t$, as follows:

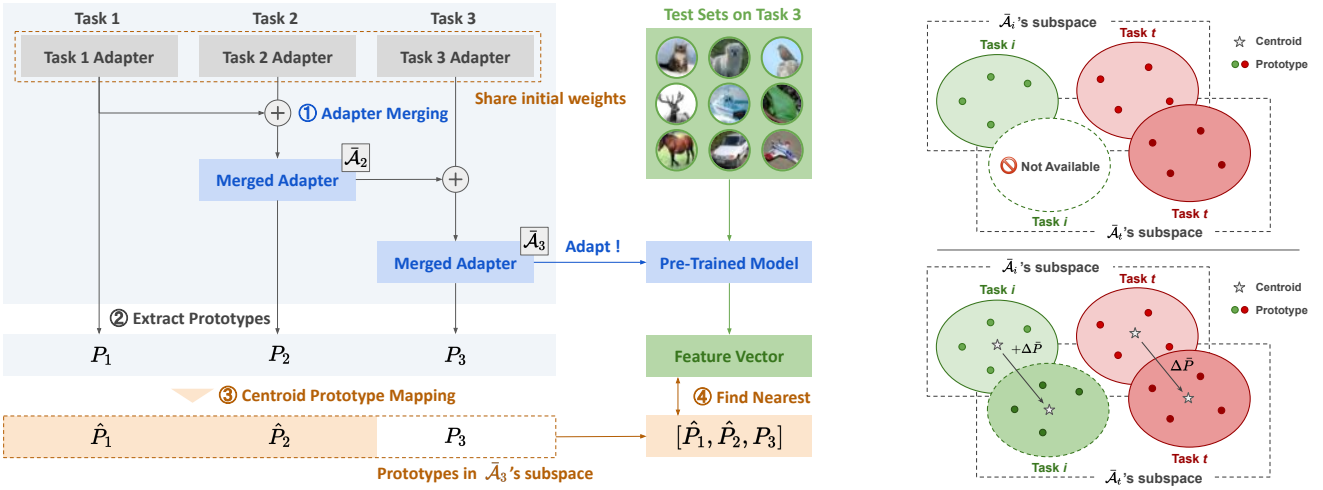
$$\boldsymbol{p}_{t,c} = \sum_{(\boldsymbol{x}_t, y_t) \in \mathcal{V}_t} \phi(\boldsymbol{x}_t)\, \mathbb{I}(y_t = c), \tag{2}$$

where $\mathbb{I}(\cdot)$ is the indicator function. Then, the prototypes are concatenated to define the prototype matrix $\boldsymbol{P}_t \in \mathbb{R}^{C_t \times d}$, where $C_t = |\mathcal{Y}_t|$ and

$$\boldsymbol{P}_t = \begin{bmatrix} \boldsymbol{p}_{t,1} & \cdots & \boldsymbol{p}_{t,C_t} \end{bmatrix}. \tag{3}$$

This calculation is performed within $\mathcal{A}_1$'s subspace [61] or across all subspaces [59]. During inference, the prototype matrices are used as the classifier weights $\boldsymbol{W} = [\boldsymbol{P}_1 \cdots \boldsymbol{P}_t] \in \mathbb{R}^{C \times d}$, where C is the total number of classes $\sum_{i=1}^{t} C_i$ learned so far. The model output f is redefined with a cosine classifier [11] as follows:

$$f(\boldsymbol{x}) = \frac{\boldsymbol{W}^\top \phi(\boldsymbol{x})}{\|\boldsymbol{W}\|_2 \, \|\phi(\boldsymbol{x})\|_2}, \tag{4}$$



(a) Overview.

(b) Centroid Prototype Mapping.

Figure 2. An Illustration of ACMap. ACMap sequentially trains an adapter for each task, starting from shared initial weights and incrementally merging them into a single adapter. In the subspace formed by the merged adapter, the prototypes for the current task are computed, while previous prototypes are updated via centroid prototype mapping.

where $\|\cdot\|_2$ denotes the ℓ_2-norm. The predicted class of $\boldsymbol{x}$ is determined as the class with the highest cosine similarity among the elements of $f(\boldsymbol{x})$. Note that prototypes for previous classes are not included in $\mathcal{V}_t$ and thus cannot be computed. In other words, in $\mathcal{A}_t$'s subspace, it is impossible to calculate $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_{t-1}$. These prototypes must be complemented with appropriate alignments.

4. ACMap: Adapter Merging with Centroid Prototype Mapping

In this paper, we propose **A**dapter Merging with **C**entroid Prototype **Map**ping (**ACMap**) for scalable CIL. Existing methods that rely on task-specific training for CIL often struggle with scalability during inference. ACMap addresses this challenge by training task-specific adapters and consolidating them into a single unified adapter via *adapter merging*. This approach allows ACMap to maintain scalability, requiring only the merged adapter during inference, while ensuring efficiency as tasks increase.

4.1. Adapter Merging in CIL

As illustrated in Figure 2a, ACMap follows a sequential process where a task-specific adapter is trained for each task, and the merged adapter $\bar{\mathcal{A}}$ is incrementally updated via adapter merging.

Adapter Merging: Adapter merging is a model merging technique that combines multiple adapters, inspired by previous work on model merging [20, 30, 51, 53]. A common approach for model merging is average merging [51], which averages the weights of multiple models with a shared initial weight. In ACMap, each task-specific adapter starts with shared initial weights $\boldsymbol{\theta}_{\text{init}}$ and undergoes task-specific training to update the weights $\boldsymbol{\theta}_t$ for each task. The merged adapter is initialized as $\bar{\mathcal{A}}_1 = \mathcal{A}_1$, and weights $\bar{\boldsymbol{\theta}}_t$ are iteratively updated via average merging:

$$\bar{\boldsymbol{\theta}}_t = \left(1 - \frac{1}{t}\right)\bar{\boldsymbol{\theta}}_{t-1} + \frac{1}{t}\boldsymbol{\theta}_t, \quad t = 2, \ldots, T. \tag{5}$$

Through this process, the adapter $\bar{\mathcal{A}}_t$ constructs a task-shared subspace that integrates the knowledge from all previous tasks. Weight averaging generally enhances both accuracy and robustness to distribution shifts compared to using a single model [24].

Towards Effective Weight Averaging: However, simply

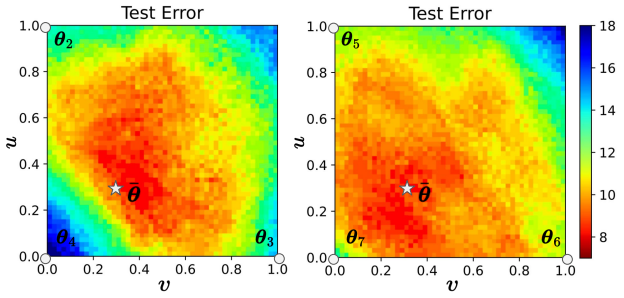


Figure 3. Visualization of the test error using linearly interpolated adapter weights $\boldsymbol{\theta} = u\boldsymbol{\theta}_{t-1} + v\boldsymbol{\theta}_t + (1-u-v)\boldsymbol{\theta}_{t+1}$, $(0 \le u, v \le 1)$ across three consecutive adapter weights $\boldsymbol{\theta}_{t-1}, \boldsymbol{\theta}_t, \boldsymbol{\theta}_{t+1}$. Test errors for the adapters $\boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4$ are shown on the left, and for the adapters $\boldsymbol{\theta}_5, \boldsymbol{\theta}_6, \boldsymbol{\theta}_7$ on the right. The star symbol indicates the average merging ($u = 1/3, v = 1/3$). Additional results are provided in Appendix G.

averaging the weights of different models does not guarantee optimal performance, particularly without proper alignment [1, 24, 25, 43]. For effective averaging, models should ideally originate from a common pre-trained model or be fine-tuned in similar regions of parameter space. This alignment helps reduce variance, enhances regularization, and ensures that interpolated weights remain within a low-loss basin, thereby supporting stable and improved performance [24].

Landscape Analysis for Adapter Merging: We analyze the loss landscape of three successive adapters, as shown in Figure 3, through the linear interpolation $\boldsymbol{\theta} = u\boldsymbol{\theta}_{t-1} + v\boldsymbol{\theta}_t + (1-u-v)\boldsymbol{\theta}_{t+1},\ (0 \leq u, v \leq 1)$, where $\boldsymbol{\theta}_{t-1}$, $\boldsymbol{\theta}_t$, and $\boldsymbol{\theta}_{t+1}$ represent the adapters trained on tasks $t-1$, t, and $t+1$, respectively. The test dataset is a combination of all three tasks: $\mathcal{T}_{t-1} \cup \mathcal{T}_t \cup \mathcal{T}_{t+1}$. This analysis shows the existence of a low-loss basin (red region), indicating that the interpolated weights exhibit consistent performance across the tasks. This basin likely forms because each adapter is initialized with the same starting parameters. This initialization leads to similar training paths in the parameter space and helps form a shared low-loss region.

Initial Weight Replacement: To further encourage the formation of a low-loss basin, we propose *initial weight replacement*. After training on the first task, a shared initial weight $\boldsymbol{\theta}_{\text{init}}$, which is typically initialized randomly, is replaced with $\boldsymbol{\theta}_1$, the parameters learned from the first task. Hence, adapters for subsequent tasks $i\ (> 1)$ are trained with $\boldsymbol{\theta}_1$ as its initial parameter. By sharing the weight of the first adapter, subsequent tasks are more likely to follow similar training paths, helping convergence within a shared low-loss region.

4.2. Prototype Mapping

In ACMap, a key challenge is an inability to compute previous prototypes within the subspace of the current adapter $\bar{\mathcal{A}}_t$, due to restricted access to data from previous tasks. Specifically, for task t, previous prototypes $\boldsymbol{P}_i(\bar{\mathcal{A}}_t), i = 1, \ldots, t-1$ cannot be computed within the current subspace, where $\boldsymbol{P}_i(\bar{\mathcal{A}}_t)$ denotes the prototypes for task i computed using the current adapter $\bar{\mathcal{A}}_t$. This limitation is because, in CIL, data from previous tasks is unavailable, as shown in Figure 2b (top).

While the unavailable prototypes can be substituted with $\boldsymbol{P}_i(\bar{\mathcal{A}}_i), i = 1, \ldots, t-1$ from previous subspaces, such substitutions may lead to an alignment problem. Figure 4a shows the cosine similarity $\text{Sim}(\boldsymbol{P}_1(\bar{\mathcal{A}}_1), \boldsymbol{P}_1(\bar{\mathcal{A}}_t))$ when substituting $\boldsymbol{P}_1(\bar{\mathcal{A}}_1)$ for $\boldsymbol{P}_1(\bar{\mathcal{A}}_t)$ where $\text{Sim}(\cdot, \cdot)$ denotes cosine similarity. This result shows that earlier task prototypes, such as from $t = 1$, shift significantly when applied to later subspaces, indicating poor alignment. Therefore, aligning prototypes within the current adapter is crucial for accurate and consistent prototype mapping. In contrast, Figure 4b shows using $\boldsymbol{P}_5(\bar{\mathcal{A}}_5)$ as a substitute for $\boldsymbol{P}_5(\bar{\mathcal{A}}_t)$ maintains high cosine similarities. This suggests that the alignment problem is not as severe. We will discuss this at the end of this section.

Algorithm 1 Centroid prototype mapping on the t-th task.

Input: Merged adapters $\bar{\mathcal{A}}_1, \ldots, \bar{\mathcal{A}}_t$, previous prototypes $\boldsymbol{P}_1(\bar{\mathcal{A}}_1), \ldots, \boldsymbol{P}_{t-1}(\bar{\mathcal{A}}_{t-1})$.
Output: Prototypes aligned within the current subspace.

$\boldsymbol{P}_t(\bar{\mathcal{A}}_t) \leftarrow$ Calculate t-th task prototype with $\bar{\mathcal{A}}_t$.
▷ Centroid prototype mapping
for $i = 1, \ldots, t-1$ **do**
  $\boldsymbol{P}_t(\bar{\mathcal{A}}_i) \leftarrow$ Calculate t-th task prototype with $\bar{\mathcal{A}}_i$.
  $\Delta\boldsymbol{p} \leftarrow \frac{1}{|\mathcal{Y}_t|}\sum_{c=1}^{|\mathcal{Y}_t|}(\boldsymbol{p}_{t,c}(\bar{\mathcal{A}}_t) - \boldsymbol{p}_{t,c}(\bar{\mathcal{A}}_i))$
  $\Delta\boldsymbol{P} \leftarrow \left[\Delta\boldsymbol{p}\ \cdots\ \Delta\boldsymbol{p}\right] \in \mathbb{R}^{d\times|\mathcal{Y}_t|}$
  $\hat{\boldsymbol{P}}_i(\bar{\mathcal{A}}_t) \leftarrow \boldsymbol{P}_i(\bar{\mathcal{A}}_i) + \Delta\boldsymbol{P}$
end for
return $\hat{\boldsymbol{P}}_1(\bar{\mathcal{A}}_t), \ldots, \hat{\boldsymbol{P}}_{t-1}(\bar{\mathcal{A}}_t)$

Centroid Prototype Mapping: The above alignment problem can be formulated as finding a mapping f: $\boldsymbol{P}_i(\bar{\mathcal{A}}_t) = f\left(\boldsymbol{P}_i(\bar{\mathcal{A}}_i)\right), i = 1, \ldots, t-1$. However, since the mapping f is generally unknown, we approximate it with an affine mapping:

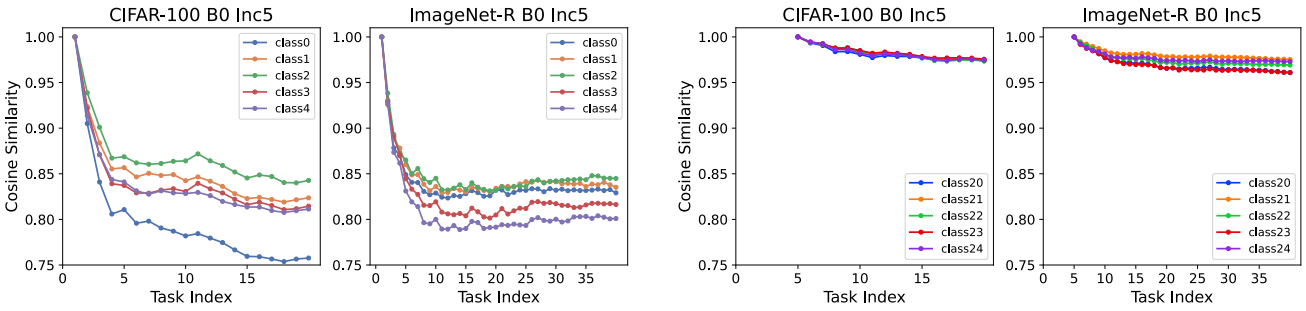
$$\boldsymbol{P}_i(\bar{\mathcal{A}}_t) \approx \boldsymbol{P}_i(\bar{\mathcal{A}}_i) + \Delta\boldsymbol{P}. \tag{6}$$

We estimate $\Delta\boldsymbol{P}$ as the difference between the centroids of the available prototypes $\boldsymbol{P}_t(\bar{\mathcal{A}}_t)$ and $\boldsymbol{P}_t(\bar{\mathcal{A}}_i)$, i.e.,

$$\Delta\boldsymbol{P} = \mathbb{E}[\boldsymbol{P}_t(\bar{\mathcal{A}}_t) - \boldsymbol{P}_t(\bar{\mathcal{A}}_i)], \tag{7}$$

where the expectation is taken over the prototypes. This approach assumes that the alignment for task t, which is computable, also applies to previous tasks $i\ (< t)$. The validity of this assumption is demonstrated experimentally. We refer to this mapping as *centroid prototype mapping*; the algorithm is detailed in Algorithm 1.

Early Stopping for Adapter Merging: As shown in Figure 4b, the alignment problem is not severe for tasks relatively close to the current one. Building on this observation, we introduce early stopping for adapter merging, which halts the merging process once the number of tasks exceeds a specified threshold. While efficient during inference, average merging increases training costs because adapters must be trained for each task. In Equation (5), as t grows, the difference between $\boldsymbol{\theta}_{t-1}$ and $\boldsymbol{\theta}_t$ becomes negligible, with the coefficient $1/t$ approaching zero. This supports the effectiveness of early stopping, as further merging becomes redundant.



(a) $\boldsymbol{P}_1(\bar{\mathcal{A}}_1)$ as a substitute for $\boldsymbol{P}_1(\bar{\mathcal{A}}_t)$.

(b) $\boldsymbol{P}_5(\bar{\mathcal{A}}_5)$ as a substitute for $\boldsymbol{P}_5(\bar{\mathcal{A}}_t)$.

Figure 4. The curve showing the differences in cosine similarity that arise when earlier task prototypes are substituted for prototypes in subsequent subspaces.

5. Experiments

We follow the the protocol in [59] to evaluate performance and inference time, and conduct an ablation study to validate our method.

5.1. Experimental Setup

Datasets: We evaluate our method on five benchmark datasets: CIFAR-100 [23] (CIFAR), CUB [47], ImageNet-R [15] (IN-R), ImageNet-A [14] (IN-A), and VTAB [56]. Dataset details are provided in Appendix A. CIFAR-100 contains 100 classes, CUB, ImageNet-R, and ImageNet-A each contain 200 classes, and VTAB consists of 50 classes. For all datasets except VTAB, the class order is randomized for each seed. For VTAB, the class order is fixed. We divide these datasets into T tasks, using the notation "B-m Inc-n", where m is the initial number of classes, and n is the number of classes added incrementally per task.

Evaluation Metrics: Following the standard protocol in CIL [36], we use two evaluation metrics: the average accuracy $\bar{A}$ across all tasks and the accuracy A_T of the final task (final accuracy).

Baselines: We compare our method with several baselines and state-of-the-art methods. The baseline is fine-tuning the pre-trained model for each task, referred to as Finetune. We also test finetuning only the adapter, referred to as Finetune Adapter. For comparison, we select CIL approaches using a pre-trained model: L2P [50], DualPrompt [49], CODA-Prompt [41], SimpleCIL [61], APER [61], and EASE [59]. Among these, SimpleCIL, APER, and EASE are prototype-based methods. SimpleCIL, a prototype-based CIL method without adapters, serves as the baseline. APER trains an adapter only on the first task and extracts features from the pre-trained model with and without the adapter. EASE, by contrast, trains separate adapters for each task and individually extracts feature vectors from each adapter.

Training Details: We follow the training settings used in [62]. For the pre-trained model, we use the ViT-B/16 model, pre-trained on ImageNet-21K [37]. Adapter training is optimized using SGD with a cosine annealing learning rate scheduler. The learning rate, batch size, and number of training epochs are set for each dataset, following the values specified in [62] (see Appendix B).

5.2. Main Results

Table 1 presents the average accuracy $\bar{A}$ and final accuracy A_T for the five benchmark datasets. The results for the comparison methods are taken from [59], while the values for ACMap (ours) represent averages from five runs. Figure 5 shows the top-1 accuracy curve during CIL. The comparison methods include SimpleCIL, APER, and EASE, all of which are prototype-based methods. The values in the figure also represent averages from five runs for all methods. Additional results are provided in Appendix E.1 and Appendix E.3.

ACMap performs comparably to or slightly better than EASE across all datasets, except VTAB B0 Inc10, and significantly outperforms APER on all datasets except CUB B0 Inc10. ACMap achieves notable improvements on domain-shifted datasets, such as IN-R, IN-A, and VTAB, compared to APER, which reuses the first adapter for all tasks. This suggests that ACMap's approach to training and merging adapters enhances domain adaptation. However, on CUB B0 Inc10, adapter learning and merging did not improve performance, as even SimpleCIL, which lacks adapters, performs comparably to both APER and EASE.

In the VTAB B0 Inc10 experiment, EASE outperforms ACMap. This difference is likely due to the VTAB setup, which includes five datasets from distinct domains, each corresponding to a separate dataset. This setup enables EASE to use five separate adapters, one per dataset, while ACMap uses a single adapter to learn across all five datasets. Additionally, a closer analysis of VTAB reveals that the fourth task has a larger dataset than the others, potentially leading to overfitting on the fourth task (see Appendix E.2). Consequently, as shown in Figure 5 (far right),

Table 1. Average accuracy $\bar{A}$ and final accuracy A_T. CIFAR refers to CIFAR-100, and IN-R/A refers to ImageNet-R and ImageNet-A. Results for the comparison methods are taken from those reported in [59]. All evaluations are conducted in an exemplar-free setting. In our methods, IR denotes initial weight replacement.

Method	CIFAR B0 Inc5		CUB B0 Inc10		IN-R B0 Inc5		IN-A B0 Inc20		VTAB B0 Inc10	
	$\bar{A}$	A_T	$\bar{A}$	A_T	$\bar{A}$	A_T	$\bar{A}$	A_T	$\bar{A}$	A_T
Finetune	38.90	20.17	26.08	13.96	21.61	10.79	24.28	14.51	34.95	21.25
Finetune Adapter [5]	60.51	49.32	66.84	52.99	47.59	40.28	45.41	41.10	48.91	45.12
L2P [50]	85.94	79.93	67.05	56.25	66.53	59.22	49.39	14.71	77.11	77.10
DualPrompt [49]	87.87	81.15	77.47	66.54	63.31	55.22	53.71	41.67	83.36	81.23
CODA-Prompt [41]	89.11	81.96	84.00	73.37	64.42	55.08	53.54	42.73	83.90	83.02
SimpleCIL [61]	87.57	81.26	92.20	86.73	62.58	54.55	59.77	48.91	85.99	84.38
APER + Adapter [61]	90.65	85.15	92.21	86.73	72.35	64.33	60.47	49.37	85.95	84.35
EASE [59]	91.51	85.80	**92.23**	86.81	**78.31**	**70.58**	**65.34**	55.04	**93.61**	**93.55**
Ours w/o IR ($L = 10$)	91.53	87.35	91.74	**87.02**	76.47	69.88	63.95	54.63	90.28	86.25
Ours w/o IR ($L = \infty$)	91.54	87.35	91.74	86.96	76.56	70.08	64.00	54.67	90.28	86.25
Ours ($L = 10$)	92.01	87.73	91.59	86.61	77.10	70.25	65.14	56.04	91.21	87.56
Ours ($L = \infty$)	**92.04**	**87.81**	91.56	86.66	77.31	70.49	65.19	**56.19**	91.21	87.56

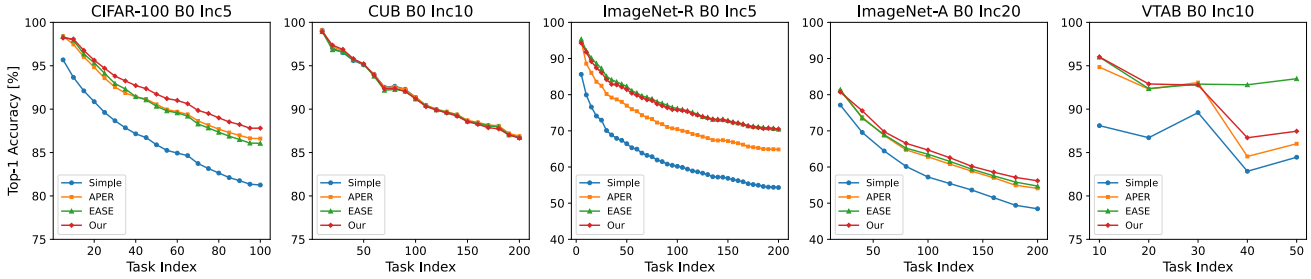


Figure 5. Top-1 accuracy curve during CIL, comparing prototype-based methods: SimpleCIL (denoted as Simple), APER, and EASE. Additional results are provided in Appendix E.1.

Table 2. Comparison of inference time for task 40 of IN-R B0 Inc5 among SimpleCIL, APER, EASE, and ACMap (ours). Time Ratio indicates how many times longer each comparative method's inference time is compared to ACMap. ACMap achieves accuracy comparable to EASE while maintaining efficient inference.

Method	Time (s)	Time Ratio
SimpleCIL [61]	22.6	$\times 0.96$
APER [61]	44.1	$\times 1.88$
EASE [59]	916.5	$\times 39.0$
ACMap (ours)	23.5	-

model accuracy declines starting from the fourth task, resulting in lower performance for ACMap than EASE.

5.3. Inference Time

Table 2 shows the inference time for task 40 of ImageNet-R B0 Inc5. The compared methods include SimpleCIL, APER, and EASE, with computational complexities of $\mathcal{O}(1)$, $\mathcal{O}(1)$, and $\mathcal{O}(T)$, respectively, where T denotes the number of tasks. ACMap achieves $\mathcal{O}(1)$ complexity by using a single adapter across tasks.

The results show that ACMap significantly outperforms EASE in terms of inference time, achieving a 39-fold speedup for 40 tasks. This speedup is particularly advantageous for real-world applications requiring many tasks and efficient inference. Compared to SimpleCIL, ACMap achieves similar inference time while improving final top-1 accuracy by over 16% (Table 1). Similarly, ACMap demonstrates comparable inference time to APER but surpasses it by over 6% in final top-1 accuracy. These results confirm ACMap's success in balancing high accuracy and inference efficiency. Additional experiments on inference time are in Appendix C.

5.4. Ablation Study

We conducted experiments on CIFAR-100 B0 Inc5 and ImageNet-R B0 Inc5 to conduct ablation studies.

Table 3. Ablation study for initial weight replacement (IR) and centroid prototype mapping (CM) with the symbol ✓ indicating the method used. Both components contribute to the improvement of performance.

IR	CM	CIFAR B0 Inc5		IN-R B0 Inc5	
		$\bar{A}$	A_T	$\bar{A}$	A_T
		90.46	86.32	75.99	69.55
	✓	91.53	87.35	76.47	69.88
✓		91.07	86.85	76.56	69.80
✓	✓	**92.01**	**87.73**	**77.10**	**70.25**

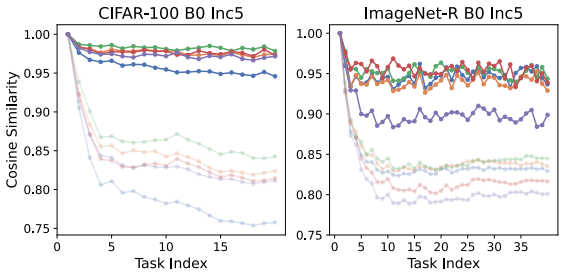


Figure 6. Cosine similarity curves of $\mathrm{Sim}(\hat{\boldsymbol{P}}_1(\bar{\mathcal{A}}_1), \boldsymbol{P}_1(\bar{\mathcal{A}}_t))$, with solid lines showing the similarity between mapped and true prototypes, and semi-transparent lines between unmapped and true prototypes. The prototypes aligned through centroid prototype mapping move closer to the true prototype in subsequent tasks. Additional results are provided in Appendix F.

Centroid Prototype Mapping and Initial Weight Replacement: We conduct ablation studies on two key components: centroid prototype mapping (CM) and initial weight replacement (IR). The results of this study are presented in Table 3. These experiments fixed the early stopping parameter L at 10. The results indicate that both CM and IR contribute to performance improvement, emphasizing their role in enhancing the model's overall effectiveness.

We further evaluate the effectiveness of centroid prototype mapping by examining how well the mapped prototypes $\hat{\boldsymbol{P}}_1(\bar{\mathcal{A}}_t)$ align with the true prototypes $\boldsymbol{P}_1(\bar{\mathcal{A}}_t)$ using cosine similarity. The true prototypes are computed using the validation datasets from previous tasks, which are unavailable in the CIL setting. The cosine similarity curve, $\mathrm{Sim}(\hat{\boldsymbol{P}}_1(\bar{\mathcal{A}}_t), \boldsymbol{P}_1(\bar{\mathcal{A}}_t))$, as adapter merging progresses, is shown in Figure 6, where $\mathrm{Sim}(\cdot\,,\cdot)$ denotes cosine similarity. The solid line represents the cosine similarity between the mapped and true prototypes, while the semi-transparent line shows the similarity between the unmapped prototypes $\boldsymbol{P}_1(\bar{\mathcal{A}}_1)$ and the true prototypes. The colors of the curves correspond to the classes from the first task. The results indicate that centroid prototype mapping effectively aligns the mapped prototypes with the true prototypes, as seen from the high cosine similarity values.

Table 4. Evaluation of the impact of the early stopping threshold L. By applying early stopping at around $L = 10$, unnecessary computations can be reduced without sacrificing model accuracy.

Threshold	CIFAR B0 Inc5		IN-R B0 Inc5	
	$\bar{A}$	A_T	$\bar{A}$	A_T
$L = 0$	91.07	86.85	76.56	69.80
$L = 5$	91.88	87.61	76.81	69.87
$L = 10$	92.00	87.76	77.09	70.25
$L = 20$	92.04	87.80	77.27	70.37
$L = \infty$	**92.04**	**87.80**	**77.31**	**70.49**

Early Stopping Threshold: We also evaluate the impact of the early stopping threshold, L, with results shown in Table 4. Since CIFAR-100 B0 Inc5 has 20 tasks, results for $L = 20$ match those for $L = \infty$. The experiments show that increasing L generally improves performance; however, the gains diminish as L increases. Furthermore, setting $L = 10$ achieves performance comparable to $L = \infty$. This indicates that early stopping avoids extra computation without sacrificing accuracy. See Appendix D for details on how the early stopping threshold L is set.

6. Conclusion

ACMap effectively addresses the challenges of CIL by retaining knowledge across tasks while scaling efficiently with an increasing number of tasks. By merging task-specific adapters into a unified adapter, ACMap ensures efficient and consistent inference over time. The centroid prototype mapping mechanism refines task representations within a shared subspace, preserving accuracy as tasks accumulate. Experimental results on five benchmark datasets demonstrate that ACMap not only matches the accuracy of state-of-the-art methods but also maintains constant inference time. This combination of competitive performance and scalability makes ACMap a promising approach for scenarios requiring efficient, scalable inference.

While ACMap demonstrates strong performance, further refinements are needed when dealing with tasks with significantly different datasets, such as those found in VTAB. A promising future direction is the concept of *adapter bank*. By selecting the most relevant adapters from a pool of candidates, the adapter bank approach could further improve ACMap's performance across diverse scenarios.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP23K24914, JP22K17962, and ROIS NII Open Collaborative Research 2025, Grant Number 251S7-22705.

References

[1] Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git Re-Basin: Merging Models modulo Permutation Symmetries. In *ICLR*, 2023. 5

[2] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert Gate: Lifelong Learning with a Network of Experts. In *CVPR*, pages 7120–7129, 2017. 2

[3] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *NeurIPS*, pages 11816–11825, 2019. 1, 2

[4] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient Lifelong Learning with A-GEM. In *ICLR*, 2019. 2

[5] Shoufa Chen, Chongjian GE, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. AdaptFormer: Adapting Vision Transformers for Scalable Visual Recognition. In *NeurIPS*, pages 16664–16678, 2022. 7

[6] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning Without Memorizing. In *CVPR*, pages 5133–5141, 2019. 2

[7] Alexey Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021. 3

[8] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. PODNet: Pooled Outputs Distillation for Small-Tasks Incremental Learning. In *ECCV*, pages 86–102, 2020. 2

[9] Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion. In *CVPR*, pages 9275–9285, 2022. 2

[10] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, pages 128–135, 1999. 1, 2

[11] Spyros Gidaris and Nikos Komodakis. Dynamic Few-Shot Visual Learning Without Forgetting. In *CVPR*, pages 4367–4375, 2018. 2, 3

[12] Xu Han et al. Pre-trained models: Past, present and future. *AI Open*, pages 225–250, 2021. 2

[13] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey. *Trans. on Machine Learning Research*, 2024. 2

[14] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural Adversarial Examples. In *CVPR*, pages 15262–15271, 2021. 6, 1

[15] Dan Hendrycks et al. The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization. In *ICCV*, pages 8320–8329, 2021. 6, 1

[16] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the Knowledge in a Neural Network. In *NIPS Workshops*, 2015. 2

[17] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*, 2022. 2

[18] Xinting Hu, Kaihua Tang, Chunyan Miao, Xian-Sheng Hua, and Hanwang Zhang. Distilling Causal Effect of Data in Class-Incremental Learning. In *CVPR*, pages 3957–3966, 2021. 2

[19] Linlan Huang et al. Class-Incremental Learning with CLIP: Adaptive Representation Adjustment and Parameter Fusion. In *ECCV*, pages 214–231, 2024. 3

[20] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *ICLR*, 2023. 4

[21] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual Prompt Tuning. In *ECCV*, pages 709–727, 2022. 1, 2

[22] James Kirkpatrick et al. Overcoming catastrophic forgetting in neural networks. *National Academy of Sciences*, pages 3521–3526, 2017. 2

[23] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 2009. 6, 1

[24] Weishi Li, Yong Peng, Miao Zhang, Liang Ding, Han Hu, and Li Shen. Deep Model Fusion: A Survey. arXiv:2309.15698, 2023. 4, 5

[25] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent Learning: Do different neural networks learn the same representations? In *NIPS Workshop*, pages 196–212, 2015. 5

[26] Zhizhong Li and Derek Hoiem. Learning without Forgetting. *IEEE TPAMI*, pages 2935–2947, 2018. 2

[27] Yan-Shuo Liang and Wu-Jun Li. InfLoRA: Interference-Free Low-Rank Adaptation for Continual Learning. In *CVPR*, pages 23638–23647, 2024. 3, 4

[28] Yaoyao Liu, Bernt Schiele, and Qianru Sun. RMM: Reinforced Memory Management for Class-Incremental Learning. In *NeurIPS*, pages 3478–3490, 2021. 2

[29] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, pages 6470–6479, 2017. 1, 2

[30] Michael S Matena and Colin A Raffel. Merging Models with Fisher-Weighted Averaging. In *NeurIPS*, pages 17703–17716, 2022. 4

[31] Mark D. McDonnell, Dong Gong, Amin Parveneh, Ehsan Abbasnejad, and Anton van den Hengel. RanPAC: random projections and pre-trained models for continual learning. In *NeurIPS*, pages 12022–12053, 2023. 3, 4

[32] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to Remember: A Synaptic Plasticity Driven Framework for Continual Learning. In *CVPR*, pages 11313–11321, 2019. 2

[33] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. In *EACL*, pages 487–503, 2021. 1

[34] Quang Pham, Chenghao Liu, and Steven Hoi. DualNet: Continual Learning, Fast and Slow. In *NeurIPS*, pages 16131–16144, 2021. 2

[35] Jathushan Rajasegaran, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Mubarak Shah. iTAML: An Incre-

mental Task-Agnostic Meta-learning Approach. In *CVPR*, pages 13588–13597, 2020. 3

[36] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental Classifier and Representation Learning. In *CVPR*, pages 5533–5542, 2017. 1, 2, 6, 3, 4

[37] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik. ImageNet-21K Pretraining for the Masses. In *NeurIPS Track on Datasets and Benchmarks*, 2021. 6, 1

[38] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NeurIPS*, pages 2994–3003, 2017. 1, 2

[39] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. In *ACM Conf. on CCS*, pages 1310–1321, 2015. 1

[40] Christian Simon, Piotr Koniusz, and Mehrtash Harandi. On Learning the Geodesic Path for Incremental Learning. In *CVPR*, pages 1591–1600, 2021. 2

[41] J. Smith, L. Karlinsky, V. Gutta, P. Cascante-Bonilla, D. Kim, A. Arbelle, R. Panda, R. Feris, and Z. Kira. CODA-Prompt: COntinual Decomposed Attention-Based Prompting for Rehearsal-Free Continual Learning. In *CVPR*, pages 11909–11919, 2023. 1, 2, 6, 7

[42] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4080–4090, 2017. 2

[43] Norman Tatro, Pin-Yu Chen, Payel Das, Igor Melnyk, Prasanna Sattigeri, and Rongjie Lai. Optimizing Mode Connectivity via Neuron Alignment. In *NeurIPS*, pages 15300–15311, 2020. 5

[44] Sebastian Thrun. Is Learning The n-th Thing Any Easier Than Learning The First? In *NeurIPS*, pages 640–646, 1995. 1

[45] Rishabh Tiwari, Krishnateja Killamsetty, Rishabh Iyer, and Pradeep Shenoy. GCR: Gradient Coreset Based Replay Buffer Selection for Continual Learning. In *CVPR*, pages 99–108, 2022. 2

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 6000–6010, 2017. 3

[47] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical report, California Institute of Technology, 2011. 6, 1

[48] Fu-Yun Wang, Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. FOSTER: Feature Boosting and Compression for Class-Incremental Learning. In *ECCV*, pages 398–414, 2022. 1, 2, 3, 4

[49] Zifeng Wang et al. DualPrompt: Complementary Prompting for Rehearsal-Free Continual Learning. In *ECCV*, pages 631–648, 2022. 1, 2, 6, 7

[50] Zifeng Wang et al. Learning to Prompt for Continual Learning. In *CVPR*, pages 139–149, 2022. 1, 2, 6, 7

[51] Mitchell Wortsman et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, pages 23965–23998, 2022. 4

[52] Jinlin Xiang and Eli Shlizerman. TKIL: Tangent Kernel Optimization for Class Balanced Incremental Learning. In *ICCV Workshops*, pages 3529–3539, 2023. 3

[53] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. TIES-Merging: Resolving Interference When Merging Models. In *NeurIPS*, pages 7093–7115, 2023. 4

[54] S. Yan, J. Xie, and X. He. DER: Dynamically Expandable Representation for Class Incremental Learning. In *CVPR*, pages 3013–3022, 2021. 1, 2, 3, 4

[55] Lu Yu, Bartlomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic Drift Compensation for Class-Incremental Learning. In *CVPR*, pages 6982–6991, 2020. 4

[56] Xiaohua Zhai et al. A Large-scale Study of Representation Learning with the Visual Task Adaptation Benchmark. arXiv:1910.04867, 2020. 6, 1

[57] Da-Wei Zhou, Qi-Wei Wang, Han-Jia Ye, and De-Chuan Zhan. A Model or 603 Exemplars: Towards Memory-Efficient Class-Incremental Learning. In *ICLR*, 2023. 1, 2, 3, 4

[58] Da-Wei Zhou, Hai-Long Sun, Jingyi Ning, Han-Jia Ye, and De-Chuan Zhan. Continual learning with pre-trained models: A survey. In *Int. Joint.Conf. on AI*, pages 8363–8371, 2024. 1, 2

[59] Da-Wei Zhou, Hai-Long Sun, Han-Jia Ye, and De-Chuan Zhan. Expandable Subspace Ensemble for Pre-Trained Model-Based Class-Incremental Learning. In *CVPR*, pages 23554–23564, 2024. 1, 2, 3, 6, 7, 4

[60] Da-Wei Zhou, Qi-Wei Wang, Zhi-Hong Qi, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Class-Incremental Learning: A Survey. *IEEE TPAMI*, pages 1–20, 2024. 2

[61] Da-Wei Zhou, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Revisiting Class-Incremental Learning with Pre-Trained Models: Generalizability and Adaptivity are All You Need. *IJCV*, 2024. 1, 2, 3, 6, 7

[62] Da-Wei Zhou, Hai-Long Sun, Han-Jia Ye, and De-Chuan Zhan. https://github.com/sun-hailong/CVPR24-Ease, 2024. 6, 2