AdaPrefix++: Integrating Adapters, Prefixes and Hypernetwork for Continual Learning

Sayanta Adhikari[1], Dupati Srikar Chandra[1], P. K. Srijith[1], Pankaj Wasnik[2], Naoyuki Oneo[2]
[1]Indian Institute of Technology Hyderabad, India [2]Sony Research India, India
{ai22mtech12005, ai20resch11004}@iith.ac.in, srijith@cse.iith.ac.in,
{Pankaj.Wasnik, naoyuki.onoe}@sony.com

Abstract

Continual learning allows systems to continuously learn and adapt to the tasks in an evolving real-world environment without forgetting previous tasks. Developing deep learning models that can continually learn over a sequence of tasks is challenging. We propose a novel method, AdaPrefix, which addresses this and empowers continual learning capability in pretrained large models (PLMs). AdaPrefix provide a continual learning method for transformer-based deep learning models by appropriately integrating the parameter-efficient methods, adapters and prefixes. AdaPrefix is an effective approach for smaller PLMs and achieves better results than state-of-the-art approaches. We further improve upon AdaPrefix by proposing AdaPrefix++, enabling knowledge transfer across the tasks. It leverages hypernetworks to generate prefixes and continually learns the hypernetwork parameters to facilitate knowledge transfer. AdaPrefix++ has a smaller parameter growth compared to AdaPrefix and is more effective and valuable for continual learning in PLMs. We performed several experiments on various benchmark datasets to demonstrate the performance of our approach for different PLMs and continual learning scenarios. Code is available on Github Link

1. Introduction

Achieving human cognitive capability has always been a goal for Machine Learning. With the introduction of Deep Learning (DL) models, the performance of machines on many complex problems like image recognition [17, 29], segmentation [51], etc., has improved drastically. Deep learning models can achieve human-level performance or even outperform humans on tasks such as image classification [7, 18, 36]. From a practical perspective, models may have to encounter tasks sequentially. For example, in autonomous driving, the vehicle may cross different sets of classes associated with different domains sequentially. It would be practically expensive to store the data from previous domains and re-train the whole model again along with a new set of classes. In such situations, it is essential for models to keep learning new tasks without forgetting the past tasks and knowledge associated with them. However, typical DL models forget all the past information as soon as they are trained on newer task data, a phenomenon known as *catastrophic forgetting* [19]. Continual Learning (CL) methods aim to address the issue of catastrophic forgetting and enable models to learn continuously over the sequence of tasks [6, 26].

Several deep Learning models such as ResNet [14], VGG [33], and EfficientNet [38] were found to be effective in solving computer vision tasks. Recently, after introducing attention-based architectures, Transformer [40] based deep learning models became widely popular in computer vision. Several computer vision tasks were more effectively modelled through transformer-based backbone architectures capable of extracting rich features [2, 9, 49].

Transformer-based large models involve a huge number of parameters and require large amounts of data to learn and solve the tasks effectively [8]. Several recent approaches use existing pretrained large transformer models (PLMs) and fine-tune them on a downstream task with limited data to perform well on the downstream tasks. Fine-tuning a full pre-trained large model is computationally expensive. There exist parameter-efficient fine-tuning (PEFT) approaches, such as adapters and prefixes, to fine-tune the PLMs for downstream tasks [16, 23, 30].

Recent work on continual learning achieves good performance on tasks using a PLM and PEFTs, such as adapters and prefixes separately [11, 12, 34, 42, 44, 46, 47, 52, 54]. Adapters and Prefixes have their advantages and disadvantages for continual learning. Prefixes are parameter efficient but sensitive to the backbone architecture's size. The performance of prefixes drops when the backbone architecture is a smaller PLM. Though Adapters were found to be effective in natural language processing, very few papers in vision have used adapters for continual learning. A drawback with adapters is that the number of parameters required is higher than that of prefixes.

We propose a novel approach *AdaPrefix*, which integrates the two PEFT methods, Adapters and Prefixes, leading to an effective approach to continual learning. However, *AdaPrefix* is based on parameter isolation and does not consider knowledge transfer among the tasks. To address this, we propose *AdaPrefix++*, which enables knowledge transfer and more effective continual learning. *AdaPrefix++* achieves knowledge transfer by generating the prefixes through a common hypernetwork across all the tasks. We train them to prevent catastrophic forgetting in hypernetworks by adding a regularisation term to the loss function. Both approaches use PLMs as the backbone architecture and tune task-specific adapters and prefixes to continuously learn the tasks. Experiments on several real-world datasets show that our approaches can provide better performance (4-7 % increment on average) than the latest baseline approaches in continual learning, especially for smaller PLMs.

The main contributions of this paper are:

- We propose integrating prefixes and adapters in Transformers for continual learning, giving rise to our *AdaPrefix* approach.
- *AdaPrefix++*, an improved version of *AdaPrefix*, is proposed to achieve knowledge transfer by generating prefixes using a shared hypernetwork. With knowledge transfer between tasks, *AdaPrefix++* can achieve less forgetting.
- We demonstrate that *AdaPrefix* and *AdaPrefix++* give better performance in the case of smaller backbone PLM architecture and comparable results in the case of larger PLMs. *AdaPrefix* is able to achieve an improvement of 2-3% whereas *AdaPrefix++* achieves an even better improvement of 4-7% in performance.

2. Related Works

There are multiple ways in which researchers have tried to achieve continual learning capability in deep learning models. The previous work can be classified into methods that use regularization [19, 21, 53], replay-based [5, 25], dynamic network growth [10, 50], or parameter isolation [27, 32, 41, 48]. Each approach comes with its own advantages and disadvantages. An elaborate description of different approaches in each category is provided in [28, 35, 43].

New parameter-efficient fine-tuning (PEFT) methods were introduced to fine-tune transformer-based pre-trained large models (PLMs) on downstream tasks. These methods have fewer parameters and provide a structured way of fine-tuning PLMs on downstream tasks, making it beneficial for continual learning. It also helps in utilizing the knowledge learnt by the PLM in a structured way during pretraining. The most widely used PEFT methods are *Adapters* and *Prefixes*. Parameter isolation methods for task-specific adapters were introduced for task-incremental CL [37]. Rather than considering independent adapters for different tasks, a set of few adapters was considered, and as a new task comes in, they distil the best adapter that approximates the current task [11]. Fewer recent works use adapters for continual learning [52, 54]. Some initial work on prompts and prefixes like L2P [45] was introduced for task-agnostic continual learning. Some of the additional work on task-agnostic setup in continual learning is DualPrompt [46], S-Prompt [44], and CODA-Prompt [34]. L2P, DualPrompt, and CODA-Prompt provided ways to construct the prompts at the input instance level from a set of prompt pools. S-Prompt maintained task-specific prompts but used K-Means to get the best-suited prompts from the prompt pool during inference. One of the recent works in prompt-based continual learning, HiDe-Prompt [42] maintains task-specific prompts and trains a task-id predictor on top of it to get the task-id during inference.

3. Background

3.1. Problem Statement

We assume that we have a set of T tasks $\{\mathcal{T}^t\}_{t=1}^{T}$ and associated datasets $\{\mathcal{D}^t\}_{t=1}^{T}$, where $\mathcal{D}^t = \{(x_k^t, y_k^t)\}_{k=1}^{N_t}$ contain N^t samples associated with task t. Here, $x_k^t \in \mathbb{R}^D$ are input samples with D features, and $y_k^t \in \{1, \cdots, C\}$ for multi-class classification and $\{+1, -1\}$ for binary classification. In a CL setting, the data associated with the tasks are not available all at once; they arrive sequentially one after the other. We need to train the model on each task successively without forgetting previously learned tasks under the constraint of limited memory. We aim to learn a model F on a sequence of tasks without catastrophic forgetting so that it can give good generalization performance on all the seen tasks.

3.2. Prefix

Prefix Tuning [23] is a fine-tuning approach that concatenates trainable vectors with the keys and values of every MHA layer of Transformers. These vectors are fine-tuned to adapt the model towards the downstream tasks. According to [23], direct training of prefixes may result in optimization instability. They propose a prefix re-parameterisation approach to address this issue and enhance the robustness of prefixes. Here, a lower-dimensional prefix (p'_l) is passed through an MLP, and then the resulting prefixes $P_l = MLP(p'_l; \theta)$ are added to the PLM. Here, P_l denotes the prefixes concatenated to l^{th} layer keys and values. p'_l generally has the same *prefix_length* (number of prefixes that are added), but the dimension of each prefix is lower than that of the actual prefix. θ are the parameters associated with the MLP used for re-parameterization.

3.3. Adapter

Pretrained large models typically have huge parameters, and it is expensive to adapt them to a downstream task through full fine-tuning of the model parameters. This is addressed by parameter-efficient fine-tuning techniques such as adapters [16]. Adapters are small neural networks that follow an auto-encoder architecture with a skip connection. They are added to the PLMs and fine-tuned to a downstream task. Adding two adapters per PLM layer, one after the Multi-Head Attention (MHA) layer and the other after Feed-Forward Network (FFN), was proposed in [16]. The performance of the PLM in the downstream task can be improved by updating relatively few parameters associated with the adapters.

An adapter block learns a function $\mathbf{A}$ that transforms the output of the transformer block specific to the downstream task. An adapter block has two sets of weights: a down-projection matrix, S, and an up-projection matrix, U. The transformation associated with it can be written as, $\mathbf{A}(\mathbf{h}) = \mathbf{h} + U^T\sigma(S^T\mathbf{h})$. Here, $\mathbf{h}$ represents the input to the adapter block, and σ represents a non-linear transformation.

4. Methodology

4.1. AdaPrefix

Adapters offer a flexible architecture, enabling quick adaptation to downstream tasks while maintaining good generalization performance. In contrast, Prefixes, although more parameter-efficient than adapters, exhibit optimal effectiveness mainly with large pre-trained language models (PLMs) [22]. To leverage the strengths of both techniques, we introduce *AdaPrefix*, a simple approach for continual learning. *AdaPrefix* combines adapters and prefixes within a PLM by integrating prefixes into the Multi-Head Attention (MHA) layer and adapters after the transformer block's Feed-Forward Network (FFN) layer. Different adapters and prefixes are associated with each task during continual learning, ensuring parameter isolation and preventing catastrophic forgetting in PLMs.

The objective is to develop a model F capable of continuous learning as tasks are presented sequentially. The proposed model F consists of three components: (a) a task agnostic PLM, represented by f with parameters Θ_p, (b) task-specific adapters A^t and Prefixes P^t with parameters denoted by ϕ^t and (c) a task-specific classifier with parameters θ^t_{cls}.

In the proposed approach, we maintain separate task-specific parameters for each task and the PLM parameters are kept frozen throughout the learning process. The task-specific parameters associated with the model consist of parameters of the adapters (U^t_i and S^t_i) and parameters associated with prefixes, (P^t_i) for each layer i of the PLM associated with task t. Compared to the PLM's actual parameters, these task-specific parameters are much less. Let,

$$\phi^t = \{\underbrace{\{S^t_i, U^t_i\}^{n_L}_{i=1}}_{\text{Adapter Params}}, \underbrace{\{P^t_i\}^{n_L}_{i=1}}_{\text{Prefix Params}}\} \tag{1}$$

where n_L represents the total number of layers in a PLM.

The proposed model F gets data x^t_k and the corresponding task ID $\mathcal{T}^t$ as input and provides the probability of class prediction, p^t_k as output, *i.e.* $\mathrm{p}^t_k = F(x^t_k, \mathcal{T}^t)$.

The input undergoes processing through the PLM backbone, along with task-specific adapters and prefixes, producing a task-specific representation. This representation is fed into the task-specific classifier head to derive predictive probabilities. The adapters and prompts, governed by parameters ϕ^t, steer the PLM to generate task-relevant representations, aiding in capturing task-specific knowledge. The task-specific parameters of *AdaPrefix* are learned by computing the *cross-entropy* loss on the training data of task t, using the output p^t_k and ground truth label y^t_k. We optimize the cross-entropy loss to learn the task-specific parameters ϕ^t and θ^t_{cls}.

$$\underset{\phi^t, \theta^t_{cls}}{\text{argmin}} \ \frac{1}{N_t} \sum_{k=1}^{N_t} [\mathcal{L}_{ce}(F(x^t_k, \mathcal{T}^t), y^t_k)] \tag{2}$$

As tasks arrive sequentially, the parameters associated with each task are optimized following Eq. (2). We utilize mini-batch stochastic gradient descent for training. Adapters converge faster compared to prefixes. We adopt the re-parameterization technique outlined in Sec. 3.2 to ensure stable optimisation and improved convergence. Here, we optimize the loss concerning the parameters of the MLP and the low-dimensional prefixes for each layer. Post-training, we store only the prefixes generated by the MLP.

4.2. AdaPrefix++

AdaPrefix is initially designed with a parameter isolation scheme, maintaining separate adapter and prefix parameters for each task. However, this approach lacks knowledge transfer capabilities across tasks, crucial for continual learning settings. To address this limitation, we propose *AdaPrefix++*, aiming to leverage knowledge transfer and enhance performance across all tasks. Inspired by the concept of hypernetworks in continual learning literature, such as [1, 4, 41], we incorporate a task-conditioned hypernetwork to generate task-specific prefixes. This hypernetwork is shared across all tasks and takes learnable task embeddings as input. It is also conditioned concerning layer embeddings to constrain the hypernetwork parameters, generating layer-specific prefix weights. *AdaPrefix++* employs a fully connected neural network as the hypernetwork, which takes the concatenation of task embedding e^t and layer embedding l_i as input to generate prefix, P^t_i.

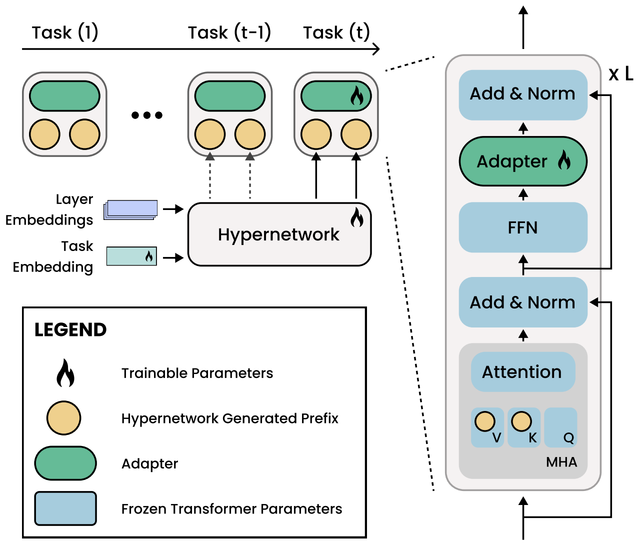

Figure 1. Shows the architecture of Adaprefix++, where adapters and hypernetwork-generated prefixes are added to a PLM for each task. The layer embeddings are fixed, and trainable task embeddings are passed through the hypernetwork to generate prefixes. Along with that, we have task-specific adapters added to the PLM.

Let the hypernetwork be denoted as $\mathcal{H}$ with its parameters as θ_h. Note that the hypernetwork parameters are the same across all the tasks. Let us consider the number of prefixes generated for a layer to be n and the dimension of each prefix (same as the embedding dimension of PLM) as d. The hypernetwork requires generating task-specific prefixes associated with keys and values, each of size $n \times d$, for each layer. To keep the hypernetwork size small, we generate them by maintaining separate tasks and layer embeddings for keys and values instead of generating both together. Let the task and layer embedding dimensions be denoted as $d_e << d$ and $d_l << d$, respectively. We consider task embedding for task t to be $e^{At} \in \mathbb{R}^{n \times d_e}$ and layer embedding of the i^{th} layer as $l_i^A \in \mathbb{R}^{n \times d_l}$, where $A \in \{K, V\}$. We use them to generate the corresponding prefixes. Considering $e^t = \{e^{K,t}, e^{V,t}\}$ and $l_i = \{l_i^K, l_i^V\}$, let's assume the prefixes of a particular layer i and task t can be generated as

$$P_i^{A,t} = \mathcal{H}([e^{A,t}, l_i^A]; \theta_h) \qquad \text{where } A \in \{K, V\} \tag{3}$$

The *AdaPrefix++* model, F consists of a backbone PLM f with its parameter as Θ_p, a set of task-agnostic parameters associated with the hypernetwork and layer embeddings, a set of task-specific parameters ϕ^t associated with the adapters and task embeddings, and a task-specific classifier head with parameters θ^t_{cls}. The layer embeddings, being task-agnostic, are kept fixed and initialized with the one-hot encoding of the corresponding layer number (Sec. 6.1 & Supplementary Sec. 4). We denote layer embeddings as $\mathrm{L} = \{l_i\}_{i=1}^{n_L}$, where n_L is the total number of layers in the PLM.

$$\phi^t = \{ \underbrace{\{S_i^t, U_i^t\}_{i=1}^{n_L}}_{\text{Adapter Parameters}}, \underbrace{\{e^t\}_{t=1}^{T}}_{\text{Task Embeddings}} \} \tag{4}$$

We can generate the prefixes for a particular task for all layers using Eq. (3), and then the generated prefixes are passed to the function F to generate prediction probabilities over the classes, $\mathrm{p}_k^t = F(x_k^t; \Theta_p, \theta_h, \phi^t, \theta^t_{cls}, \mathrm{L})$. Fig. 1 provides the architecture details of the *AdaPrefix++*, where a hypernetwork can be seen to generate the task-specific prefixes.

For the *AdaPrefix++* training, if we use the objective function defined for *AdaPrefix* (i.e. Eq. (2)), we could learn the task-specific parameters ϕ^t. However, the task agnostic parameters, particularly hypernetwork parameter θ_h, are fine-tuned on the last task, leading to catastrophic forgetting on earlier tasks. Consequently, it fails to properly generate the prefixes for the earlier sequence tasks. To mitigate catastrophic forgetting in hypernetwork, we use regularization on the outputs of the hypernetwork. We define θ_h as parameters of hypernetwork while training on task t. To overcome forgetting in hypernetworks, we add a regularization term along with the task-specific loss (Eq. (2)). When training for task t, the regularization term can be written as

$$\mathcal{L}_R^t = \frac{1}{n'_L} \sum_{k=1}^{t-1} \sum_{i=1}^{n_L} \left\| \mathcal{H}([e^k, l_i]; \hat{\theta}_h) - \mathcal{H}([e^k, l_i], \theta_h) \right\|_2^2. \tag{5}$$

where $n'_L = (t-1)n_L$. To compute the regularization loss $\mathcal{L}_R^t$ for the current task, we store the hypernetwork weights learned from previous tasks as $\hat{\theta}_h$. The hypernetwork parameters θ_h are optimized using a combination of cross-entropy and regularization losses. This regularization ensures that the hypernetwork, when learning the t^{th} task, generates prefixes consistent with those produced by the hypernetwork trained on all previous tasks, given the same task and layer embeddings. This regularization is motivated by knowledge distillation loss used in the domain of CL [3, 24]. We associate a regularization constant, λ, which controls the stability-plasticity trade-off in the model.

The final optimization of the task-specific and task-agnostic parameters is obtained by minimizing $\phi^t, \theta_h, \theta^t_{cls}$ on the joint loss $\mathcal{L}^t_{ce} + \mathbb{1}_{t \neq 1}.\lambda \mathcal{L}^t_R$. Here, $\mathcal{L}^t_{ce}$ is the cross-entropy loss between the predicted labels and ground-truth labels for task t, and $\mathcal{L}^t_R$ is the regularization loss associated with task t as defined above. θ^t_{cls} are the parameters associated with task-specific classifier heads. Further training details are provided in Algorithm 1.

AdaPrefix++ provides forward transfer as the knowledge learned by the hypernetwork in the previous task can provide a better starting point for the model in predicting prefixes. *AdaPrefix++* only generates prefixes using hypernetwork and maintains separate task-specific adapter parameters. If we intend to generate adapter parameters using a hypernetwork, the parameters of the hypernetwork become pretty large, and the parameters associated with the hypernetwork itself become close to the backbone network. Consequently, training the hypernetwork requires large data and becomes computationally expensive. This has practical limitations under the memory constraint set up of continual learning.

Algorithm 1 *AdaPrefix++* Training Algorithm

procedure TRAINING($\{\mathcal{T}^t : \mathcal{D}^t\}_{t=1}^{T}, F, \lambda$)
  Initialize θ_h.
  Initialize L. ▷ One-Hot encoding used
  for $t \leftarrow 1$ **to** T **do**
    Initialize ϕ^t, θ_{cls}^t, e^t
    for $(x_k^t, y_k^t) \in \mathcal{D}^t$ **do** ▷ Batch training is used
      Use e^t, θ_h, L to generate P^t for all layers
      Optimize $\{\phi^t, \theta_h, \theta_{cls}^t\}$ using $\mathcal{L}$
      $(\mathcal{L} = \mathcal{L}_{ce}^t + \mathbb{1}_{t\neq 1}.\lambda\mathcal{L}_R^t)$ (Eq. (5))
    end for
    $\hat{\theta}_h \leftarrow \theta_h$ ▷ To compute $\mathcal{L}_R$ for next task
    Store ϕ^t, θ_{cls}^t.
  end for
end procedure

AdaPrefix and *AdaPrefix++* can be adapted to the class incremental setup (CIL) where the task identities are not provided during inference time. In this case, we infer the task identity using the entropy score. Given an input x without task identities, we first get the probability distribution over the classes for all the seen task-ids, and we calculate the entropy over all those probability distributions for each task-id. We infer the task-id as the one in which the model has the lowest entropy, i.e. the one where the model has the highest confidence in predicting the class, $\hat{t} = \arg\min_{k\in[T]} \mathbb{E}\left(F(x; \Theta_p, \theta_h, \phi^k, \theta_{cls}^k, \mathrm{L})\right)$. Here, $\mathbb{E}(p) = -\sum_i p_i \log(p_i)$ is the entropy of the output probability distribution p, $[T] = \{1, \cdots, T\}$ and $\hat{t}$ denotes predicted task-id. Now, using $\hat{t}$ and x, we will perform inference as in task incremental learning (Supplementary Sec. 1). We follow a similar approach for inference in a domain incremental learning (DIL) scenario.

5. Experimental Setup

5.1. Baselines

We conducted experiments on benchmark datasets in task incremental learning (TIL) and class incremental learning (CIL) scenarios. We considered two dataset setups: **Split-CIFAR100**, where CIFAR100 [20] is divided into 10 tasks with 10 classes each, and **Split-ImageNet-R**, where Imagenet-R [15] is divided into 10 tasks with 20 classes each. Additionally, we experimented with the **CDDB-Hard** [44] dataset in the domain incremental learning (DIL) scenario, a continual deepfake detection benchmark.

We also explored the impact of different backbone sizes of the Vision Transformer (ViT) [9] architecture on our approaches. We considered four sizes: ViT-L (307M parameters), ViT-B (86M parameters) [9], DeiT-S (21M parameters), and DeiT-T (5M parameters) [39].

For comparison, we included traditional continual learning methods such as Experience Replay (ER) [31] and Elastic Weight Consolidation (EWC) [19], along with recent adapters and prompt-based approaches like LAE-Adapter [13], LAE-Prefix [13], L2P [45], DualPrompt [46], CODA-Prompt [34], and S-Prompt [44]. To ensure a fair comparison in TIL, we modified the classifier head to be task-specific. We also included HiDe-Prompt [42], the current state-of-the-art prompt-based approach for PLMs. In our experiments, we adapted HiDe-Prompt to use the provided task ID directly instead of a task ID predictor for a TIL comparison.

5.2. Metrics

We employ several widely used metrics from the literature [6,25]. The first metric is *average test accuracy* (higher is better), denoted as $\mathcal{A}_T$ This is computed as the average of test accuracies for all tasks at the end of training. The second metric, *forward transfer* (higher is better), assesses how training continually benefits the model. Since task-specific parameters like task embeddings and adapter parameters exist, evaluating the model's performance requires training it until a specific task. Therefore, we compute the forward transfer metric as $FWT = \frac{1}{T}\sum_{t=1}^{T}(\mathcal{A}_t - \mathcal{B}_t)$, where $\mathcal{B}_t$ represents the test accuracy of the random initialized model when trained independently on the t^{th} task and $\mathcal{A}_t$ represents test accuracy on t^{th} task of a model continually trained till t^{th} task.

6. Results

The results in Tab. 1 and Tab. 2 demonstrate that the proposed approaches *AdaPrefix* and *AdaPrefix++* outperform all the other SOTA approaches in the domain of CL. Tab. 1 demonstrates that our approaches outperform other state-of-the-art methods for both CIFAR100 and ImageNet-R datasets in CIL and TIL scenarios. This superior performance is observed across all backbone PLMs, specifically DeiT-S and DeiT-T. In the case of DeiT-T in the TIL scenario, our approach can achieve around a 5-6 % increase in accuracy for both datasets. In the case of ViT-B for the TIL scenario, our approach can increment by 3%.

From Tab. 1, CIL scenario, we can observe that for both the datasets, our approach achieved around 4-7 % increment in performance. We also observe the TIL and CIL performance of our approaches, *AdaPrefix* and *AdaPrefix++*, are almost similar in all the cases, which presents the stability

Table 1. Average Accuracy(↑) comparison of our approach with other state-of-the-art continual learning methods for task incremental (TIL) and class incremental (CIL) for different benchmark datasets.

Scenarios	**TIL**				**CIL**			
Methods	**ViT-L**	**ViT-B**	**DeiT-S**	**DeiT-T**	**ViT-L**	**ViT-B**	**DeiT-S**	**DeiT-T**
				CIFAR100				
FT-seq-frozen	83.52	80.25	62.48	55.99	17.94	17.59	15.28	12.85
EWC	85.25	81.49	62.94	56.68	64.12	59.49	52.94	48.78
ER	89.21	85.23	69.08	59.45	76.25	71.53	69.08	59.45
Adapter	97.64	96.45	93.20	92.00	96.79	94.99	92.11	90.10
LAE-Prefix	97.01	96.89	91.15	86.69	89.75	89.25	86.15	84.69
LAE-Adapter	97.51	96.89	92.44	89.24	90.01	89.59	86.45	80.51
L2P	94.95	94.35	88.33	84.02	84.20	83.06	78.21	71.02
DualPrompt	95.12	94.90	90.14	85.16	87.10	86.60	81.14	72.46
CODA-Prompt	96.00	95.52	89.86	85.35	88.56	86.94	81.86	70.35
S-Prompt	96.99	96.85	92.45	85.90	89.41	88.81	84.45	79.90
Hide-Prompt	**98.14**	97.50	95.25	87.00	93.25	92.61	89.25	85.11
AdaPrefix	97.82	97.10	94.50	92.92	96.92	96.11	93.20	89.75
AdaPrefix++	**98.14**	**97.76**	**95.88**	**93.24**	**96.99**	**96.81**	**94.98**	**91.90**
				IMAGENET-R				
FT-seq-frozen	64.58	64.15	54.78	50.17	15.93	14.63	13.98	10.11
EWC	64.25	63.12	51.10	49.22	52.10	49.11	45.28	41.02
ER	71.11	69.10	60.12	58.10	57.11	49.55	49.52	45.25
Adapter	87.87	87.50	84.00	72.50	81.25	81.11	77.22	62.10
LAE-Prefix	83.11	81.89	73.14	70.01	78.84	77.55	72.11	60.32
LAE-Adapter	88.41	88.14	85.10	72.51	80.11	79.07	74.51	61.94
L2P	76.99	75.65	69.25	61.25	71.99	71.65	62.25	59.25
DualPrompt	76.54	75.99	69.54	63.95	72.54	71.79	62.54	59.95
CODA-Prompt	82.21	79.03	72.40	66.25	75.21	75.03	65.40	61.25
S-Prompt	79.24	77.68	72.21	66.21	75.24	74.68	63.21	60.21
Hide-Prompt	87.54	85.06	79.21	69.99	77.25	76.45	72.21	63.99
AdaPrefix	88.31	87.95	84.45	73.72	81.95	83.13	79.95	65.31
AdaPrefix++	**89.79**	**89.51**	**85.25**	**74.24**	**84.39**	**84.26**	**79.81**	**67.80**

Table 2. Average Accuracy Results (↑) on **CDDB-Hard** deep-fake detection dataset in a domain incremental setting (DIL)

Methods	**ViT-L**	**ViT-B**	**DeiT-S**	**DeiT-T**
EWC	51.10	50.59	40.25	29.34
ER	74.01	73.90	65.24	51.24
LAE-Adapter	75.11	74.01	69.24	64.99
L2P	62.10	61.28	56.47	42.14
DualPrompt	61.99	61.39	57.01	42.11
CODA-Prompt	66.14	65.22	59.14	47.14
S-Prompt	75.12	74.51	70.25	63.10
AdaPrefix++	**78.45**	**77.06**	**75.21**	**67.12**

of our approaches, and degradation in performance is minimal when moving from TIL to CIL.

From Tab. 2, we can observe that our approach *AdaPrefix++* outperforms all the other SOTA approaches. Our approach achieves a 3-5% increment in performance. Overall, we can observe that our approach provides better accuracy for all the PLMs. This shows that our approach can provide better performance irrespective of the size of the backbone PLMs.

AdaPrefix, as a parameter isolation approach, achieves zero forgetting. In the case of *AdaPrefix++* also, we achieved a very low amount of forgetting (around 1%) compared to other SOTA approaches. This proves that our approach provides better accuracy even with less forgetting. The Tables in the paper contain average accuracy over 3 random initializations. More descriptive results with variances and different task orders are provided in the supplementary material (Supplementary Sec. F.2-F.5). Apart from this, we

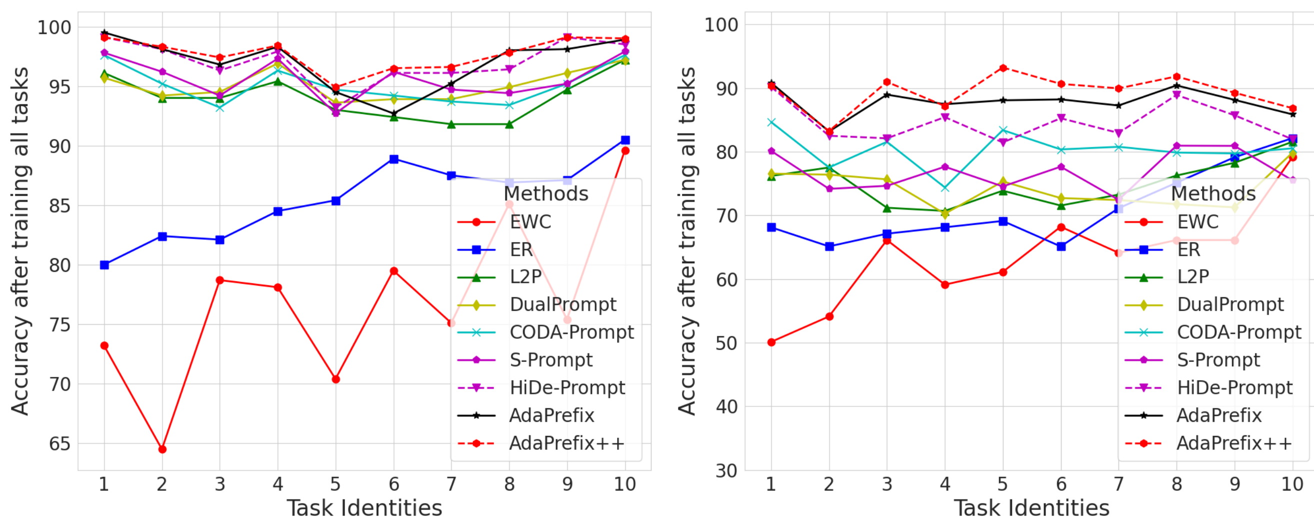

Figure 2. This figure shows the task-wise performance of all the approaches after training on all the tasks is over. The plot on the left side is for the CIFAR100 dataset, and on the ViT-B backbone, the plot on the right side is for the ImageNet-R dataset with the ViT-B backbone. These plots clearly show that *AdaPrefix++* also has good task-specific performance.

Table 3. Average Accuracy($\mathcal{A}_T$)(↑) and Forward Transfer(FWT)(↑)comparison for TIL scenario, to show how the choice of different components is required for our approach

Datasets	**Methods**	**ViT-L**		**ViT-B**		**DeiT-S**		**DeiT-T**	
		$\mathcal{A}_T$	FWT	$\mathcal{A}_T$	FWT	$\mathcal{A}_T$	FWT	$\mathcal{A}_T$	FWT
Cifar100	Prefix	96.57	0.00	96.26	0.00	90.56	0.00	85.07	0.00
	Adapter	97.64	0.00	96.45	0.00	93.20	0.00	92.00	0.00
	AdaPrefix	97.82	0.00	97.10	0.00	94.50	0.00	92.92	0.00
	Hnet+Prefix	96.86	**0.09**	97.22	0.15	91.70	**0.18**	87.19	**0.18**
	AdaPrefix++	**98.14**	0.03	**97.76**	**0.16**	**95.88**	0.01	**93.24**	0.02
ImageNet-R	Prefix	82.45	0.00	81.04	0.00	73.14	0.00	70.01	0.00
	Adapter	87.87	0.00	87.50	0.00	84.00	0.00	72.50	0.00
	AdaPrefix	88.31	0.00	87.95	0.00	84.45	0.00	73.72	0.00
	Hnet+Prefix	83.12	0.10	84.89	0.50	78.40	**0.71**	63.95	**0.66**
	AdaPrefix++	**89.79**	**0.12**	**89.51**	**0.51**	**85.25**	0.48	**74.24**	0.43

have also provided results with a longer domain incremental sequence in the supplementary Tab. 14.

Forward Transfer (FWT) Tab. 3 shows that *AdaPrefix++*'s Hypernetwork-based prefixes improve performance and enable forward transfer. Unlike parameter isolation-based approaches (Adapter, Prefix, *AdaPrefix*), *AdaPrefix++* utilizes past knowledge for better accuracy and forward transfer. It outperforms Hnet+Prefix in forward transfer on ViT-B and ViT-L for ImageNet-R.

AdaPrefix++ demonstrates good average accuracy across models and datasets and strong task-wise performance. Fig. 2 illustrates that for CIFAR100 on ViT-B, *AdaPrefix++* outperforms baselines in most task-specific accuracies, indicating stability and minimal forgetting when learning new tasks.

6.1. Ablation Study

Importance of different components Tab. 3 presents an extensive ablation study demonstrating how different components enhance our model's performance. We compare independent Adapters, Prefixes, their combination (*AdaPrefix*), hypernetwork-generated prefixes (Hnet+Prefix), and our approach *AdaPrefix++*. Tab. 3 shows how each component plays a role in the increment of performance of our model.

Importance of training Hypernetwork Tab. 4 show that training the hypernetwork is important as it helps facilitate better forward transfer, increasing performance. Due to the regularization term, the Hypernetwork parameters

Table 4. The Table shows the importance of training Hypernetwork over all the tasks. A#1 = Freeze Hypernetwork after 1^{st} task, A#2 = Train Hypernetwork with $\mathcal{L}_R$ for all tasks

Settings	CIFAR100		ImageNet-R	
	$\mathcal{A}_T$	FWT	$\mathcal{A}_T$	FWT
A#1	97.01	0.01	87.99	0.09
A#2	**97.76**	**0.16**	**89.51**	**0.39**

change sufficiently to generate optimal parameters for all the tasks without forgetting.

Table 5. Effect of $\mathcal{L}_R$ on the performance (average accuracy ($\uparrow$)) of the model is provided in this table. Different values of regularization constant λ are considered.

Datasets	CIFAR100		ImageNet-R	
λ Values	**ViT-B**	**DeiT-S**	**ViT-B**	**DeiT-S**
1	95.11	94.12	81.98	79.51
0.1	**96.75**	94.22	**83.31**	79.59
0.01	96.22	94.85	83.01	**80.05**
0.001	95.14	**95.03**	82.51	78.98
0	93.14	93.12	80.45	75.98

Effect of Hyperparameter λ: The hyperparameter λ plays a crucial role in deciding the stability and plasticity of the model. We searched for the best value of this hyperparameter in the set of $\{0, 0.001, 0.01, 0.1, 1\}$. Tab. 5 provides a detailed overview of how different values of λ affect different models and datasets.

Table 6. This table shows the performance ($\mathcal{A}_T$) of *AdaPrefix++* for different initialization of the layer embeddings.

Layer Embeddings	TIL		CIL	
	ViT-B	**DeiT-S**	**ViT-B**	**DeiT-S**
CIFAR100				
full learnable	97.04	**96.04**	96.28	**95.37**
first learnable	97.71	95.69	96.46	95.17
fixed random	97.36	95.94	96.53	95.18
fixed sine	97.64	95.58	96.39	94.77
fixed One-Hot	**97.76**	95.88	**96.81**	94.98
IMAGENET-R				
full learnable	89.5	85.31	83.78	80.48
first learnable	88.84	85.27	82.73	80.59
fixed random	89.09	**85.46**	83.51	**80.64**
fixed sine	89.22	85.09	**84.41**	79.55
fixed One-Hot	**89.51**	85.25	84.26	79.81

Different initialization of layer embeddings Layer ID Embeddings provide layer-specific information to the hypernetwork for generating prefixes. These embeddings are task-agnostic and capture constant layer information across tasks. Tab. 6 demonstrates how different initialization methods affect model performance. We explored various learnable and fixed embeddings, varying performance across backbone models and datasets. Fixed One-Hot initialization performed best for ViT-B and competitively in other cases. For consistency, we used fixed One-Hot initialization across all experiments. Detailed initialization methods are provided in the supplementary material (Supplementary Sec. C).

Combination of prefix length and adapter reduction factor *AdaPrefix* and *AdaPrefix++* use prefixes and adapters with tunable hyperparameters. We performed a grid search using prefix lengths $\{10, 15, 20, 30, 40, 50, 60, 80, 100\}$ and adapter reduction factors $\{2, 4, 8, 16, 32\}$. A prefix length of 15 and an adapter reduction factor of 8 yielded optimal performance across most experiments. We used these values for all subsequent experiments. Further details are available in the supplementary section (Supplementary Sec. E).

Importance of layer placement The effectiveness of *AdaPrefix* and *AdaPrefix++* depends on their placement within the PLM layers. Our experiments revealed that adding them to initial layers improved TIL but significantly decreased CIL performance. Conversely, placement in final layers reduced the TIL-to-CIL performance drop but had low overall performance. We found that applying the methods to all layers yielded optimal results, with alternating layer placement performing similarly well. For consistency, we applied our approach to all PLM layers throughout our experiments. Detailed experimental results are present in the supplementary material (Supplementary Sec. D).

7. Conclusion

This study introduces two novel approaches to mitigating catastrophic forgetting and improving knowledge transfer in continual learning settings. The proposed methods *AdaPrefix* and *AdaPrefix++*, both surpass the current state-of-the-art. Our method demonstrates strong performance even when paired with smaller pretrained large models. Even though our approach focuses primarily on task-incremental learning, we consistently outperform current state-of-the-art methods in both class incremental and domain incremental settings. In particular, *AdaPrefix++* achieves effective knowledge transfer with minimal forgetting, highlighting our approach's potential impact and generalizability.

Acknowledgment

We acknowledge the funding support from Sony Research India (SRI) and infrastructure support from the Japan International Cooperation Agency (JICA).

References

[1] Dhanajit Brahma, Vinay Kumar Verma, and Piyush Rai. Hypernetworks for continual semi-supervised learning. *arXiv preprint arXiv:2110.01856*, 2021. 3

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 1

[3] Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 9516–9525, 2021. 4

[4] Dupati Srikar Chandra, Sakshi Varshney, PK Srijith, and Sunil Gupta. Continual learning with dependency preserving hypernetworks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2339–2348, 2023. 3

[5] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018. 2

[6] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021. 1, 5

[7] Tiancan Deng. A survey of convolutional neural networks for image classification: Models and datasets. In *2022 International Conference on Big Data, Information and Computer Network (BDICN)*, pages 746–749, 2022. 1

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1, 5

[10] Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. Dytox: Transformers for continual learning with dynamic token expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9285–9295, 2022. 2

[11] Beyza Ermis, Giovanni Zappella, Martin Wistuba, Aditya Rawal, and Cédric Archambeau. Continual learning with transformers for image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3774–3781, 2022. 1, 2

[12] Beyza Ermis, Giovanni Zappella, Martin Wistuba, Aditya Rawal, and Cedric Archambeau. Memory efficient continual learning with transformers. *Advances in Neural Information Processing Systems*, 35:10629–10642, 2022. 1

[13] Qiankun Gao, Chen Zhao, Yifan Sun, Teng Xi, Gang Zhang, Bernard Ghanem, and Jian Zhang. A unified continual learning framework with general parameter-efficient tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11483–11493, 2023. 5

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[15] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *ICCV*, 2021. 5

[16] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019. 1, 3

[17] Yu Huang and Yue Chen. Autonomous driving with deep learning: A survey of state-of-art technologies. *arXiv preprint arXiv:2006.06091*, 2020. 1

[18] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022. 1

[19] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017. 1, 2, 5

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 5

[21] Timothée Lesort, Andrei Stoian, and David Filliat. Regularization shortcomings for continual learning. *arXiv preprint arXiv:1912.03049*, 2019. 2

[22] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021. 3

[23] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021. 1, 2

[24] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. 4

[25] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017. 2, 5

[26] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, 2022. 1

[27] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. 2

[28] Martin Mundt, Yongwon Hong, Iuliia Pliushch, and Visvanathan Ramesh. A wholistic view of continual learning with deep neural networks: Forgotten lessons and the

bridge to active and open world learning. *Neural Networks*, 160:306–336, 2023. 2

[29] Celard P. A survey on deep learning applied to medical images: from simple artificial neural networks to generative models. *Neural Computing and Applications*, 2023. 1

[30] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*, 2020. 1

[31] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019. 5

[32] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pages 4548–4557. PMLR, 2018. 2

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[34] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11909–11919, 2023. 1, 2, 5

[35] Shagun Sodhani, Mojtaba Faramarzi, Sanket Vaibhav Mehta, Pranshu Malviya, Mohamed Abdelsalam, Janarthanan Janarthanan, and Sarath Chandar. An introduction to lifelong supervised learning. *arXiv preprint arXiv:2207.04354*, 2022. 2

[36] Mohsen Soori, Behrooz Arezoo, and Roza Dastres. Artificial intelligence, machine learning and deep learning in advanced robotics, a review. *Cognitive Robotics*, 3:54–70, 2023. 1

[37] Tejas Srinivasan, Ting-Yun Chang, Leticia Pinto Alva, Georgios Chochlakis, Mohammad Rostami, and Jesse Thomason. Climb: A continual learning benchmark for vision-and-language tasks. *Advances in Neural Information Processing Systems*, 35:29440–29453, 2022. 2

[38] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 1

[39] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021. 5

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1

[41] Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019. 2, 3

[42] Liyuan Wang, Jingyi Xie, Xingxing Zhang, Mingyi Huang, Hang Su, and Jun Zhu. Hierarchical decomposition of prompt-based continual learning: Rethinking obscured suboptimality. *Advances in Neural Information Processing Systems*, 36, 2024. 1, 2, 5

[43] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. 2

[44] Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning. *Advances in Neural Information Processing Systems*, 35:5682–5695, 2022. 1, 2, 5

[45] Zhen Wang, Liu Liu, Yiqun Duan, Yajing Kong, and Dacheng Tao. Continual learning with lifelong vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 171–181, 2022. 2, 5

[46] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, pages 631–648. Springer, 2022. 1, 2, 5

[47] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 139–149, 2022. 1

[48] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020. 2

[49] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021. 1

[50] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017. 2

[51] Ying Yu, Chunping Wang, Qiang Fu, Renke Kou, Fuyu Huang, Boxiong Yang, Tingting Yang, and Mingliang Gao. Techniques and challenges of image segmentation: A review. *Electronics*, 12, 2023. 1

[52] Marko Zeman, Jana Faganeli Pucer, Igor Kononenko, and Zoran Bosnić. Superformer: Continual learning superposition method for text classification. *Neural Networks*, 161:418–436, 2023. 1, 2

[53] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pages 3987–3995. PMLR, 2017. 2

[54] Wentao Zhang, Yujun Huang, Tong Zhang, Qingsong Zou, Wei-Shi Zheng, and Ruixuan Wang. Adapter learning in pretrained feature extractor for continual learning of diseases. *arXiv preprint arXiv:2304.09042*, 2023. 1, 2