ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240718

Learning After Learning: Positive Backward Transfer in Continual Learning

Wern Sen Wong [a,*], **Yun Sing Koh**[a] **and Gillian Dobbie**[a]

[a]School of Computer Science, The University of Auckland, New Zealand

Abstract. Continual Learning (CL) methods enable models to learn new tasks without forgetting previously learned ones. Catastrophic Forgetting (CF) occurs when the parameters of a neural network are updated for a new task, causing the model to lose performance on tasks it has previously learned. To mitigate CF, parameter isolation methods use a "task mask" to allocate a subset of weights to each task; these weights are typically frozen to preserve task performance. However, frozen weights can limit positive backward transfer, which is the beneficial reuse of knowledge from new tasks to improve the accuracy of previously learned tasks. To address this gap, we introduce LEarning AFter learning (LEAF), a novel CL method that enables positive backward transfer by dynamically updating frozen task masks based on gradient updates that signal sufficient backward knowledge transfer. This mechanism allows for selective integration of new knowledge without sacrificing previously acquired knowledge. Our experiments show that LEAF surpasses existing state-of-the-art methods in terms of accuracy while maintaining comparable memory and runtime efficiencies. Moreover, it outperforms other backward transfer techniques in improving the accuracy of a prioritized task. Our code is available at https://github.com/wernse/LEAF.

1 Introduction

Continual Learning (CL) is a learning approach that involves learning a sequence of tasks. An ideal continual learner is designed to facilitate positive forward and backward transfer of knowledge, using past knowledge to learn new tasks and new knowledge to improve learning on older tasks. However, this is difficult due to the challenge of Catastrophic Forgetting (CF), where neural networks often lose knowledge from previous tasks when learning new ones, leading to a significant performance decrease [22]. Approaches to mitigate CF include regularization methods [40, 8, 32] that penalize significant weight changes associated with previous tasks, and memory-based methods [3, 25] that leverage past data samples to maintain old knowledge. Another approach is parameter isolation, which involves creating distinct subsets of neural network weights for each task. These subsets, often referred to as "subnetworks", are represented through a "task mask" which is a sparse, binary overlay that isolates the weights used for a task. Once a task is learned, these weights are frozen to prevent changes when learning new tasks [26, 35, 10].

However, while parameter isolation methods effectively mitigate CF, they come with the limitation of preventing positive backward knowledge transfer, which is the beneficial reuse of knowledge from new tasks to improve the accuracy of previously learned tasks. Intuitively, careful modifications of the learned model of old tasks may further improve learning performance, especially when the new task shares knowledge similar to the old tasks. This gap leads to an essential question: How can we improve the learned model of older tasks by facilitating backward transfer? Continuous learning systems have an important need to improve an older task constantly. For example, driver assistance systems [15] must adapt continuously to changing conditions and can improve when learning new tasks such as identifying the current time of day [19] or weather conditions [38]. Studies in replay and regularization methods have shown that positive backward transfer for specific older tasks is possible when replaying past knowledge [16, 2]. However, positive backward transfer in parameter isolation methods remains difficult as previously learned tasks remain frozen to prevent CF. In our work, we demonstrate that the dynamic selection of task mask updates, combined with the use of a replay buffer, can facilitate positive backward transfer.

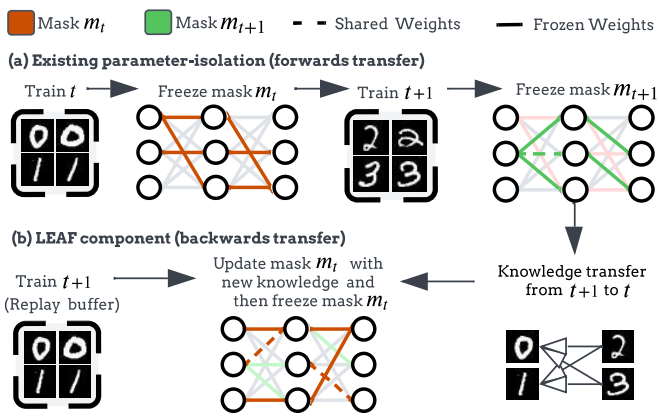

Figure 1. An illustration of the gap between existing parameter isolation methods and LEAF: (a) Existing methods freeze task mask m_{t_i} preventing updates after learning new tasks (b) LEAF component, updates the task mask m_{t_i} with new knowledge after learning task t_{i+1}.

To address the limitation of positive backward transfer in parameter isolation methods, we introduce LEarning AFter learning (LEAF), a novel method that dynamically updates an older task mask when new knowledge will be beneficial. LEAF introduces a flexible mechanism for updating older task masks with newly acquired knowledge. This update occurs when there is a significant shift in gradient updates, signaling the potential for improving the accuracy of an older task through backward transfer. When a significant change is detected, LEAF dynamically updates the older task mask

* Email: wwon129@aucklanduni.ac.nz

using data from the replay buffer and the latest network weights. This update allows the frozen task to benefit from subsequent learning. For instance, as shown in Figure 1, updating the frozen task mask for recognizing zeroes and ones with recent learnings from the task involving twos and threes can significantly improve performance for the frozen task. This update mechanism represents a paradigm shift from existing parameter isolation methods, which freeze task masks and thus prevent updates. Our LEAF approach allows for dynamic updates, enabling positive backward knowledge transfer.

Our contributions are threefold:

- We introduce a novel parameter-isolation technique that dynamically updates an older task mask when newly learned knowledge is beneficial. This approach allows the update of an older task, allowing LEAF to enable positive backward transfer for an older task while maintaining comparable memory and runtime.
- We define a specific backward transfer scenario for CL, which aims to improve the accuracy of a chosen older task by storing past task data. This scenario allows LEAF to demonstrate the potential accuracy improvements for a critical older task, which is difficult in a CL setting [16].
- Our evaluation of LEAF highlights that our proposed method outperforms state-of-the-art methods on 4 out of 5 benchmark datasets, both in terms of accuracy and backward transfer for the chosen older task.

2 Related Work

CL strategies encompass several directions, notably regularization, rehearsal, and parameter isolation methods. Regularization approaches [40, 8] incorporate constraints to minimize the adjustment of weights for previously learned tasks. Rehearsal strategies [29, 3, 25] employ a storage buffer to store past task data and replay a subset during learning to preserve past knowledge.

Parameter Isolation Methods target CF by assigning unique network parameter subsets to each task, primarily utilizing parameter freezing to maintain task knowledge. Most works have focused on defining *mask-based methods*, where a learned mask for individual weights or groups of weights is used to freeze or constrain specific parameters selectively. Piggyback [21] uses a binary mask on the weights of a pre-trained model to create different subnetworks; an additional overhead is introduced of 1 bit per network parameter for each task. Kang et al. [10] introduced Winning SubNetworks (WSN), which sequentially learns and selects an optimal subnetwork for each task using an *accumulate binary mask*. WSN updates only weights not selected in previous tasks, resulting in a task-specific subnetwork. Another direction is Soft-masking of Parameter-level Gradient flow (SPG) [11], which uses the mask concept. However, it applies progressive update penalties on important parameters instead of a binary mask to prevent forgetting. Ada-QPacknet [26] combines adaptive pruning with bit width reduction to compress the models across tasks efficiently by reducing the bit-width of the weights format. Our methodology diverges by iteratively updating frozen task masks when new information is detected, facilitating both forward and backward knowledge transfer through a dynamic task mask.

Knowledge Transfer Methods have mainly focused on the CF issue, with limited exploration in knowledge transfer across tasks. Divided into experience-replay and orthogonal-projection categories, experience-replay methods such as A-GEM [2] aim to balance task learning by replaying combinations of old and new task data. Orthogonal-projection methods offer a different approach; Gradient Projection Memory (GPM) [31] stores the bases of the subspaces spanned by old task data and projects the new gradients on the directions orthogonal to these subspaces. Similarly, Trust Region Gradient Projection (TRGP) [17] proposes a scaled weight projection to facilitate the forward knowledge transfer from related old tasks to the new task while updating the model based on orthogonal gradient projection. Notably, the ContinUal learning method with Backward knowlEdge tRansfer (CUBER) [16] marks a significant advancement by enabling positive backward transfer, where it selectively projects gradients derived from the task's entire training data to determine when to update the knowledge of old tasks that are positively related to the current task. In light of their contributions, these methods depend on storing gradients or data, which is important for training subsequent tasks and attempting to improve backward transfer. Inspired by storing past data, we combine this with the strengths of parameter isolation methods. To the best of our knowledge, our paper is the first to attempt to improve backward transfer in parameter isolation models using a replay buffer.

Full Access to Past Data Methods facilitates knowledge transfer with full access to all past data. Such works as Model Zoo [28] and application areas such as in robotics [37] advocate for the accessibility of full access task data to improve accuracy and knowledge transfer between tasks. Our scenario aims to selectively store sufficient past data from a chosen task to enable backward transfer.

3 Backwards Transfer Learning Scenario

In the field of CL, going beyond CF and facilitating backward transfer is a significant challenge [16], with substantial implications for robotics [37] and driver assistance systems [15]. These applications depend on improving certain operations with new task knowledge. For example, precise environmental perception is crucial for safe navigation in driver assistance systems. The environmental perception task can benefit from knowledge gained from related tasks like discerning the time of day [19] and weather conditions [38].

Existing CL models attempt to improve backward transfer store past concepts such as CUBER [16], which stores task gradients, and A-GEM [2], which stores a sample of past task data. However, the amount of backward transfer is limited by the past data they can use to improve a learned task. On the other hand, models with unrestricted access to past task data [28, 37] face challenges in data storage. Our approach proposes a novel scenario, allowing a chosen crucial task to store its full set of data in the buffer, thereby providing the potential to improve positive backward transfer without overwhelming system resources. Using the knowledge from new tasks, our method seeks not only to mitigate CF but also to improve the performance of a chosen crucial task.

3.1 Problem Statement

We encounter T tasks sequentially presented to a learner in a supervised learning framework. Each task t is associated with a dataset $D_t = \{x_{i,t}, y_{i,t}\}_{i=1}^{n_t}$, where $x_{i,t}$ and $y_{i,t}$ denote the input features and the corresponding label for n_t instances. We employ a neural network $f(\cdot;\theta)$, with weights θ, to learn these tasks by minimizing the objective:

$$\theta^* = \underset{\theta}{\text{minimize}} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}(f(x_{i,t};\theta), y_{i,t}), \tag{1}$$

with $\mathcal{L}(.,.)$ as the loss function, typically cross-entropy. Access to D_t is limited to its respective learning phase, while some CL methods rehearse a portion of past datasets. The task identity is known during both training and testing.

Given the propensity for neural networks to be over-parameterized to accommodate future tasks [9], identifying efficient subnetworks becomes feasible. The optimal binary mask m_t^* for task t can be defined as:

$$m_t^* = \underset{m_t \in \{0,1\}^{|\theta|}}{\text{minimize}} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}(f(x_{i,t}; \theta \odot m_t), y_{i,t}) - C, \quad (2)$$

where $\odot$ signifies element-wise multiplication, C is the loss for task t, and the model's capacity c is significantly less than the size of θ. Section 4 highlights the process to derive m_t^* using adjustable weight scores s, optimizing each task's loss collectively.

A subnetwork f_t derived from the neural network f for task t, is outlined by a binary mask m_t applied to the network's weights θ. This mask m_t specifies the active weights for task t, effectively enabling only a subset of weights. For each mask element $m_{t,i}$, a value of 1 activates the corresponding weight θ_i for task t, while 0 deactivates it. The functionality of the subnetwork f_t is defined as:

$$f_t(x; \theta, m_t) = f(x; \theta \odot m_t), \quad (3)$$

where $\odot$ represents element-wise multiplication. The output is a neural network configuration that operates with a subset of weights defined by the mask m_t, thus optimizing the network for the specific task t.

3.2 Optimizing an older chosen task's accuracy

To optimize an older chosen task, we use a memory buffer $\mathcal{M}$ to store instances from the chosen task t_c to replay in the future. We aim to find a binary attention mask $m_{t_c}^*$ that minimizes the loss on the chosen task. The data for task t_c in the memory buffer is denoted as $\mathcal{M}_{t_c} = \{x_{i,t_c}, y_{i,t_c}\}_{i=1}^{n_{t_c}}$, where x_{i,t_c} represents the input features, y_{i,t_c} is the label the model is aiming to predict, and n_{t_c} represents the number of input and label pairs for task t_c in $\mathcal{M}$.

The objective, using the model parameters $\theta_{t'}$ after learning tasks up to task t' (where $t' > t_c$), is expressed as:

$$m_{t_c}^* = \underset{m_{t_c} \in \{0,1\}^{|\theta|}}{\text{minimize}} \frac{1}{n_{t_c}} \sum_{i=1}^{n_{t_c}} \mathcal{L}(f(x_{i,t_c}; \theta_{t'} \odot m_{t_c}), y_{i,t_c}) \quad (4)$$

where x_{i,t_c} and y_{i,t_c} represent the input and label for the chosen task t_c, sourced from the memory buffer $\mathcal{M}$, and $\theta_{t'}$ denotes the neural network parameters trained up to task t'.

4 LEarning After learning algorithm

We describe LEarning AFter learning (LEAF) in Algorithm 1, a backward transfer method that uses task masks, gradient changes, and a replay buffer to enable positive backward transfer, exploit newly learned knowledge, and maintain comparable runtime and memory. Figure 2 presents the overview of the LEAF process, illustrating the evolution of task masks and the chosen task mask within the neural network.

Unlike state-of-the-art forget-free methods such as PackNet [20] and WSN [10], which prevent CF by freezing previous task masks without considering potential benefits from backward knowledge transfer, LEAF adds an additional step to dynamically determine if an older task should be unfrozen and be carefully updated with knowledge from the new task.

LEAF operates in two main steps. Initially, for each task, LEAF identifies a high-accuracy task mask and updates only the weights that have not been trained on the previous tasks. After training, the model freezes the task mask parameters so that the previous task is immune to the CF. Subsequently, LEAF detects if the new knowledge can improve the accuracy of a previously learned task. This detection is determined by identifying significant shifts in the gradient updates for the chosen task. Upon detecting a significant shift, the chosen task mask is dynamically updated with data from the replay buffer using the latest weights, which encapsulate new knowledge from subsequent tasks. This approach ensures that the mask benefits from the most up-to-date network weights for backward transfer by incorporating knowledge after the task was initially frozen.

4.1 Task mask generation

In this stage, the goal is to find a high-accuracy subnetwork for each task while preventing weight updates to any existing subnetworks, thus mitigating CF. Following the methodology of Kang et al. [10], a task-specific mask m_t for task t is derived. This mask selectively activates weights that improve the accuracy of the current task. Each weight θ is evaluated and assigned a score s throughout the training process. The mask m_t is formed by selecting the top $c\%$ of weights from each layer based on these scores, where c represents a fixed percentage of each layer's capacity. This approach results in a sparse binary mask, m_t, defined explicitly for task t.

To optimize model weights θ and task mask m_t using the task's dataset D_t, we use an objective $\mathcal{L}(\cdot)$ to minimize θ and s as follows:

$$\underset{\theta, s}{\text{minimize}} \, \mathcal{L}(\theta \odot m_t; D_t). \quad (5)$$

Creating subnetworks introduces two challenges: first, updating θ across new tasks risks altering weights critical for older tasks; second, the zero-gradient indicator function complicates updating the weight scores s. For the first challenge, our approach involves selective weight updates, focusing only on the weights not selected for earlier tasks. We use an accumulated binary mask M_{t-1} for task t as follows:

$$M_{t-1} = \cup_{i=1}^{t-1} m_i, \quad (6)$$

where m is a mask for a chosen task and accordingly update θ:

$$\theta \leftarrow \theta - \eta \left(\frac{\partial L}{\partial \theta} \odot (1 - M_{t-1}) \right), \quad (7)$$

where η represents an optimization algorithm, $\frac{\partial L}{\partial \theta}$ denotes the gradient update, and $\odot$ signifies the element-wise multiplication of the gradient with the binary mask for task t. For the second issue, we use the Straight-through Estimator [27] in the backward pass to avoid the derivatives of the indicator function, allowing for the update of the weight score s.

$$s \leftarrow s - \eta \left(\frac{\partial L}{\partial s} \right). \quad (8)$$

The weight scores s are used to select weights for the subnetwork.

4.2 Updating the chosen mask

Updating a chosen older task mask m_{t_c} is a key component of LEAF. This process aims to integrate relevant new knowledge into an older

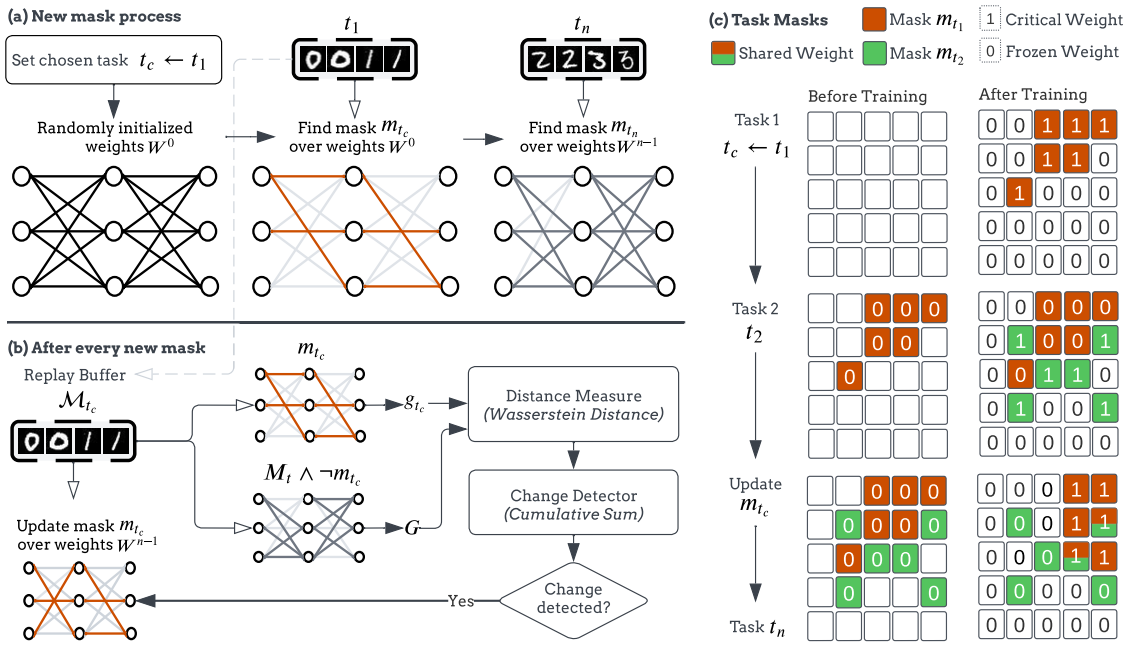

Figure 2. Overview of LEAF: (a) The model initially identifies task-specific masks for incoming tasks. When the chosen task is learned, the chosen task data is stored within a replay buffer for future replay. (b) Updates are triggered by measuring the gradient divergence between the chosen mask m_{t_c} and the accumulated binary mask of all other previous tasks M_t. (c) A visual representation is provided to illustrate the evolution of task masks. After task 2, m_{t_c} is updated with new knowledge, resulting in shared weights between m_{t_c} and task 2's mask m_{t_2}.

task, improving its accuracy while keeping other task masks frozen to prevent CF. To facilitate this, we use the weight scores in Equation 8 to selectively update m_{t_c}. The weight scores signal which weights are most relevant for the task mask based on the latest knowledge acquired by the network.

To execute the update, m_{t_c} is temporarily unfrozen, allowing it to carefully update to new knowledge while keeping the other task masks frozen. We adapt Equation 7 to the following:

$$\theta \leftarrow \theta - \eta \left(\frac{\partial \mathcal{L}}{\partial \theta} \odot (1 - M_t \wedge \neg m_{t_c}) \right). \tag{9}$$

Here, M_t denotes the accumulated binary mask for all tasks learned up to task t, and m_{t_c} is the binary mask for the chosen task being updated. To enable updates to m_{t_c}, we use $\wedge\neg$ to separate m_{t_c} from the accumulated mask M_t.

The chosen task mask m_{t_c} is updated by using the weight score s described in Equation 8, which guides the selection of new weights for the chosen task based on the updated knowledge in the network.

4.3 When to Transfer

In this stage, determining when to update an older task mask m_{t_c} is important to prevent negative backward transfer and maintain comparable runtime. To determine when there is significant knowledge to improve a previous task, we monitor shifts in the distribution of the gradient updates to assess whether the model has acquired diverse new knowledge suitable for transfer to the chosen task. Such shifts may signal a knowledge difference between newly learned concepts and the existing understanding of the chosen task.

To evaluate these distribution shifts quantitatively, we compute the Wasserstein distance [30] between the gradients of the aggregated binary mask, excluding the chosen task's mask, and the gradients specific to the chosen task. A substantial distance between these gradients suggests a notable divergence [18].

To detect changes, we calculate the gradient changes as follows:

$$g_{t_c} = \nabla_\theta \mathcal{L}(\theta \odot m_{t_c}; b_t), \quad G = \nabla_\theta \mathcal{L}(\theta \odot M'; b_t), \tag{10}$$

with b_t a batch from the chosen task, m_{t_c} the mask for the chosen task, and M' the accumulated binary mask excluding the chosen task. The Wasserstein distance between these gradients is:

$$W(g_{t_c}, G) = \left(\int_0^1 \left| F_{g_{t_c}}^{-1}(q) - F_G^{-1}(q) \right|^p dq \right)^{1/p}, \tag{11}$$

Here, the Wasserstein distance $W(g_{t_c}, G)$ calculates the differential gradient distributions between the chosen task mask g_{t_c} and the cumulative mask G. The calculation measures the minimum cost to transform one distribution into the other, reflecting differences in their gradient distributions.

For change detection, we utilize the Cumulative Sum (CUSUM) algorithm [24], known for its adeptness at recognizing minor yet consistent data shifts [5]. This method assists with identifying when newly acquired knowledge significantly differs from previous task knowledge, indicating backward transfer potential. Cumulative changes are determined by:

$$\begin{aligned} C_n^+ &= \max(0, C_{n-1}^+ + Q_t - k), \\ C_n^- &= \max(0, C_{n-1}^- - Q_t - k) \end{aligned} \tag{12}$$

where k is the allowance parameter that provides a buffer against minor changes, and Q_t is the Wasserstein distance between the gradients G_{t_c} and G after learning task t. An update to the chosen task mask is considered if $C_n^+ > h$ or $C_n^- < -h$ where h is a threshold that signals a significant cumulative change.

5 Experiments

Our experiments address the following research questions: 1) How does training subsequent tasks affect backward transfer on the ini-

Algorithm 1 LEAF (LEarning AFter learning)

input dataset of task $\{D_t\}_{t=1}^{T}$, model weights θ, score weights s, binary mask $\mathrm{M}_0 = 0^{|\theta|}$, layer-wise capacity c, chosen task buffer $\mathcal{M}$, chosen task t_c
for task $t = 1, \ldots, T$ **do**
  for batch $b_t \sim D_t$ **do**
    if $t_c == t$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{b_t\}$ ▷ Store chosen task
    end if
    Get mask m_t of the top-$c\%$ scores s at each layer
    Compute $\mathcal{L}(\theta \odot m_t; b_t)$
    $\theta \leftarrow \theta - \eta\big(\frac{\partial \mathcal{L}}{\partial \theta} \odot (1 - \mathrm{M}_{t-1})\big)$ ▷ Weight update
    $s \leftarrow s - \eta\big(\frac{\partial \mathcal{L}}{\partial s}\big)$ ▷ Weight score update
  end for
  $M_t \leftarrow M_{t-1} \cup m_t$ ▷ Accumulate binary mask
  if $t >= t_c$ **then**
    $m_{t_c} \leftarrow m_t$
    $M' \leftarrow M_t \wedge \neg m_{t_c}$ ▷ Accumulated mask without m_{t_c}
    $g_{t_c} \leftarrow \nabla_\theta \mathcal{L}(\theta \odot m_{t_c}; b_t)$
    $G \leftarrow \nabla_\theta \mathcal{L}(\theta \odot M'; b_t)$
    $W \leftarrow \mathrm{DistanceMeasure}(g_{t_c}, G)$ ▷ Compute distance
    if ChangeDetector(W) **then**
      for batch $b_{t_c} \sim \mathcal{M}$ **do**
        Update m_{t_c} of top-$c\%$ scores s at each layer
        Compute $\mathcal{L}(\theta \odot m_{t_c}; b_{t_c})$
        $\theta \leftarrow \theta - \eta\big(\frac{\partial \mathcal{L}}{\partial \theta} \odot (1 - \mathrm{M}')\big)$
        $s \leftarrow s - \eta\big(\frac{\partial \mathcal{L}}{\partial s}\big)$
      end for
      $M_t \leftarrow \mathrm{M}' \cup m_{t_c}$ ▷ Add updated chosen mask
    end if
  end if
end for

tially learned task in LEAF? 2) How does varying the selection of the chosen task t_c impact the backward transfer in LEAF when different tasks are chosen? 3) How does varying the buffer size allocation affect the accuracy of the chosen initial task in LEAF? 4) What are the trade-offs between backward transfer and the training time and memory requirements when using LEAF compared to other CL methods?

5.1 Experimental Setup

In our experimental setup, LEAF is presented with a sequence of learning tasks. Once a task is learned, its training data is discarded, apart from the *chosen task* t_c where the data is stored in a replay buffer. After learning all tasks, each task model is evaluated using its test data. In training each task, we use its validation set to decide when to stop training. We consider task-incremental CL with a multi-head configuration for all experiments in the paper. We follow the experimental setups in recent works [10, 31, 6].

Hyperparameter setting: We selected $t_c = 0$ to investigate the potential for positive backward transfer on the initially learned task. For LEAF, the sparsity parameter c is set to 0.3 for CIFAR100 and CIFAR100-SC, while it is set to 0.1 for MiniImageNet, TinyImageNet, and the 5 Dataset to learn each dataset. The allowance value k is determined through a grid search on the validation set. It varies according to the scale of features in each dataset: 3 for MiniImageNet, 6 for CIFAR100 and the 5 Dataset, 12 for CIFAR100-SC, and 16 for TinyImageNet. Please refer to the supplementary material [34] for more detailed information on the hyperparameters used.

Datasets and architecture: We evaluate our CL algorithm on five continual learning benchmark datasets. A *task* represents an image classification task for a given group of objects. Following the CL approaches [4], each dataset's training and testing instances are evenly split into each task and each class. These datasets are CIFAR100 [12], a visual object dataset split into 10 tasks with 10 different classes each; CIFAR100-SC [39], which segments the CIFAR100 dataset into 20 tasks based on 20 superclasses, with each task comprising 5 semantically related classes; MiniImageNet [33], a variant of ImageNet [13] with 20 tasks, each containing 5 different classes; TinyImageNet [14], another ImageNet variant with 40 tasks, each including 5 different classes; and 5-Dataset [31], a combination of five distinct vision datasets (CIFAR-10 [12], MNIST [7], SVHN [23], FashionMNIST [36], and notMNIST [1]), with the classification problem in each dataset treated as a separate task. We trained our models on an Nvidia Tesla A100 GPU 40GB. For evaluating CIFAR100 and CIFAR100-SC, we employ a 5-layer AlexNet, with 3 convolution layers and 2 fully connected layers, similar to Lin et al. [17]. For MiniImageNet, TinyImageNet, and the 5-Dataset, we use a scaled-down version of ResNet-18, similar to Kang et al. [10].

Compared Baselines. We compare our approach against one upper limit baseline and eight CL baselines. Our evaluation includes a variety of benchmarks, such as Multitask Learning (**MTL**), which joint learns all tasks, a baseline for average task accuracy. We include four knowledge transfer techniques: **GPM** [31] retains early task inputs' subspace, using orthogonal gradient projections for updates. **TRGP** [17], and **CUBER** [16] propose advanced weight and gradient management strategies for knowledge transfer and model adaptation across tasks. CUBER is particularly relevant for its focus on positive backward transfer. **A-GEM** [2] restricts learning on new tasks by utilizing gradients from old tasks' data. **SupSup** [35] and **WSN** [10] explore the use of task-specific binary masks and show the effectiveness of weight reuse and is the current state-of-the-art for parameter-isolation models. **SPG** [11] applies regularization based on the parameter's importance to the task. **Ada-QPacknet** [26] combines pruning with compression techniques through quantization.

Performance Metrics Following Lin et al. [17], we use accuracy (ACC) to measure the average test classification accuracy of all tasks. Our scenario focuses on a single chosen task setting. We introduce a metric called Backward Transfer of the Chosen Task (BWT_{t_c}). This metric is designed to measure a learned task's accuracy change after learning new tasks, defined as:

$$\mathrm{BWT}_{t_c} = \frac{1}{T - t_c} \sum_{i=t_c+1}^{T} (R_{i,t_c} - R_{t_c,t_c}). \tag{13}$$

Here, T is the total task count, t_c is the chosen task, R_{i,t_c} is the model's accuracy on task t_c after sequentially learning up to task i, and R_{t_c,t_c} is the accuracy on task t_c directly after it has been learned. The BWT_{t_c} metric ranges from -100% to 100%, measuring the accuracy change on task t_c after subsequent tasks have been learned. Negative BWT_{t_c} values indicate a decrease in the chosen task's accuracy, which indicates a loss of previously learned information due to interference from newly learned tasks. In contrast, positive values indicate increased accuracy, suggesting that learning new tasks has contributed to understanding task t_c.

5.2 Exploring Backward Transfer in LEAF (RQ1)

This section assesses how training on subsequent tasks affects backward transfer on the initially learned task. The evaluation results in Table 1 show LEAF's effectiveness in facilitating positive backward transfer, improving BWT_{t_c} over other parameter-isolation and memory-based approaches in four out of five datasets.

Table 1. The BWT_{t_c} and ACC, each with standard deviations, over 5 runs for each dataset.

Metric	Method	CIFAR100	CIFAR100-SC	TinyImageNet	5 Dataset	MiniImageNet
BWT_{t_c}	A-GEM (ICLR 19)	-5.41 ± 0.46	-4.86 ± 0.86	-7.16 ± 1.25	-8.61 ± 0.50	-7.61 ± 1.04
	GPM (ICLR 21)	-0.22 ± 0.14	-1.60 ± 0.78	-1.90 ± 0.64	-4.52 ± 1.28	-4.00 ± 1.02
	TRGP (ICLR 22)	-0.25 ± 0.16	-0.44 ± 0.26	-0.50 ± 0.30	-0.18 ± 0.32	0.50 ± 0.27
	CUBER (Neurips 22)	0.34 ± 0.12	0.40 ± 0.23	0.38 ± 0.31	-0.24 ± 0.09	1.07 ± 0.68
	SPG (ICML 23)	-1.30 ± 0.53	-1.20 ± 1.02	-1.60 ± 0.84	-2.54 ± 0.95	-1.14 ± 0.75
	SupSup (Neurips 20)	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.00 ± 0.00**	0.00 ± 0.00
	WSN (ICML 22)	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.00 ± 0.00**	0.00 ± 0.00
	Ada-QPackNet (ECAI 23)	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.00 ± 0.00**	0.00 ± 0.00
	LEAF (ours)	**0.95 ± 0.28**	**2.44 ± 0.65**	**3.99 ± 0.79**	-0.27 ± 0.18	**4.51 ± 1.12**
ACC	Multi-task	79.75 ± 0.38†	61.00 ± 0.20†	77.10 ± 1.06†	93.44 ± 0.12†	69.46 ± 0.62†
	A-GEM (ICLR 19)	62.54 ± 1.02	51.60 ± 0.84	57.51 ± 1.15	84.59 ± 1.32	58.31 ± 1.02
	GPM (ICLR 21)	72.08 ± 0.45	57.60 ± 0.37	61.09 ± 0.62	91.00 ± 0.52	61.80 ± 1.35
	TRGP (ICLR 22)	75.40 ± 0.26	58.47 ± 0.51	62.81 ± 0.82	92.78 ± 0.43	64.36 ± 1.12
	CUBER (Neurips 22)	75.71 ± 0.25	60.28 ± 0.57	67.62 ± 0.70	93.13 ± 0.27	63.34 ± 0.83
	SPG (ICML 23)	68.34 ± 0.72	56.75 ± 0.39	60.75 ± 0.79	92.28 ± 1.86	70.75 ± 0.39
	SupSup (Neurips 20)	75.22 ± 1.00	60.43 ± 0.63	58.62 ± 1.74	91.39 ± 0.85	70.43 ± 1.32
	WSN (ICML 22)	77.84 ± 0.36	64.15 ± 0.44	71.74 ± 1.15	92.91 ± 0.04	72.73 ± 1.55
	Ada-QPackNet (ECAI 23)	74.10 ± 0.65	59.54 ± 0.96	71.00 ± 1.02	**94.10 ± 0.60**	69.21 ± 0.43
	LEAF (ours)	**77.91 ± 0.41**	**64.24 ± 0.47**	**71.84 ± 1.27**	92.85 ± 0.12	**72.84 ± 1.72**

Note: † denotes results reported from Dang et al.[6].

Specifically, LEAF achieves improvements in BWT_{t_c} on CIFAR100, CIFAR100-SC, TinyImageNet, and MiniImageNet, with gains of 0.95%, 2.44%, 3.99%, and 5.21% respectively. These results show LEAF's capability to dynamically update m_{t_c} and integrate newly learned knowledge effectively. Compared to CUBER, a state-of-the-art backward transfer technique, LEAF exhibits higher BWT_{t_c} improvements by margins of 0.31%, 1.84%, 5.67%, and 4.14%. These improvements may be attributed to LEAF's update mechanism, which incorporates new network weights into the chosen task mask. However, LEAF has a slight decrease in BWT_{t_c} on the 5-Dataset, similar to CUBER and TRPG, which includes fewer interrelated tasks, a larger volume of instances per task, and a smaller number of tasks which suggest a challenge for backward transfer.

Regarding overall accuracy, LEAF leads in 4 out of 5 datasets, with gains of 0.1%, 0.11%, 0.11%, and 0.35% in ACC over SupSup, WSN, and Ada-QPackNet. These slight increases in ACC indicate that LEAF not only maintains accuracy on all other tasks but also improves the backward transfer on the chosen task.

5.3 Impact of Varied Task Selection t_c (RQ2)

This study examines the impact of varying the chosen task on backward transfer in LEAF. We hypothesize that early tasks are likely to exhibit higher accuracy gains due to their limited exposure to the knowledge from other tasks. These tasks can benefit more from integrating insights gained from subsequent tasks. In contrast, mid-sequence tasks, which already integrate knowledge from earlier tasks, may experience a saturation effect. Thus, additional learning could lead to diminishing returns, as shown by minor improvements in accuracy from newly acquired knowledge. To empirically test this hypothesis, we selected four early tasks (t_0 to t_3), four middle tasks (t_9 to t_{12}), and four late tasks (t_{15} to t_{19}) as the chosen task independently. We measure the accuracy after learning each subsequent task until all remaining tasks were learned, as shown in Figure 3.

Our experimental results indicate that early tasks chosen as t_c in LEAF improved accuracy as additional tasks were learned, outperforming all other baselines. For example, tasks t_0, t_1, t_2, and t_3 initially show no change in accuracy after the first few tasks are learned, as the update mechanism has not been activated. However, as more tasks were learned, there was an increase of 2.95%, 1.69%, 2.34%, and 2.16%, respectively. In contrast, middle and late sequence tasks do not show any accuracy changes, implying a lack of beneficial knowledge transfer from subsequent tasks. These results highlight LEAF's capability to improve positive backward transfer effectively for earlier tasks while preventing CF for later tasks.

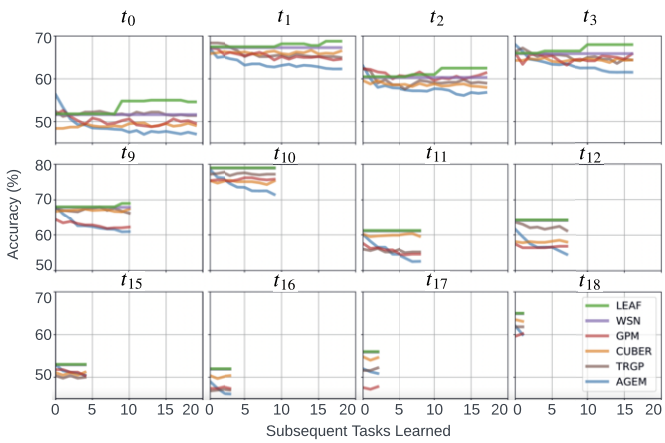

Figure 3. Accuracy of chosen tasks as subsequent tasks are learned. i.e. if $t_c = t_{12}$ and there are 20 tasks, then 7 tasks can be learned (CIFAR100-SC)

5.4 Ablation Study

We conducted an ablation study to evaluate the impact of the task mask change detector within LEAF. For this purpose, we introduced LEAF_{NCD}, which does not include a mechanism to evaluate when to update the chosen task mask based on significant changes in gradient divergence. Instead, LEAF_{NCD} updates the task mask each time the model learns a new task. Table 2 presents the comparative accuracy results for both LEAF and LEAF_{NCD} across five datasets. The results indicate that selectively updating the task mask based on gradient divergence improves the backward transfer of

the chosen task. Specifically, LEAF outperforms LEAF$_{NCD}$ on CIFAR100, CIFAR100-SC, TinyImageNet, 5 Dataset, and MiniImageNet, achieving BWT$_{t_c}$ improvements of 0.35%, 0.18%, 0.35%, 0.13%, and 0.46% respectively. This update mechanism improves backward transfer and reduces training runtime by only updating when substantial new knowledge is detected.

Table 2. BWT_{t_c} and standard deviation values for LEAF$_{NCD}$

	LEAF$_{NCD}$	LEAF
CIFAR100	0.60 ± 0.32	0.95 ± 0.28
CIFAR100-SC	2.26 ± 0.59	2.44 ± 0.65
TinyImageNet	3.53 ± 1.07	3.99 ± 1.12
5 Dataset	-0.40 ± 0.22	-0.27 ± 0.18
MiniImageNet	3.86 ± 0.68	4.51 ± 0.79

5.5 *Effect of Buffer Size on Task Accuracy (RQ3)*

This section examines how variations in the replay buffer size affect the accuracy of the chosen task t_c in LEAF, compared to LEAF$_{NU}$, which lacks a task mask update mechanism. We systematically vary the buffer size to include 100%, 80%, 60%, 40%, and 20% of the total instances available for task t_c, where 100% stores the complete set of training instances for task t_c. The variants of LEAF are designated as LEAF$_{100}$, LEAF$_{80}$, LEAF$_{60}$, LEAF$_{40}$, and LEAF$_{20}$, representing the percentage of the total training instances stored in the buffer. This experiment is run on four datasets: TinyImageNet, MiniImageNet, CIFAR100, and CIFAR100-SC, as shown in Table 3.

The findings show that as the buffer size decreases, the accuracy across all datasets decreases. When using LEAF, the accuracy of the chosen task exceeds that of LEAF$_{NU}$ at 80% buffer size for CIFAR100, 60% for CIFAR100-SC, and 40% for both TinyImageNet and MiniImageNet. The reduction in buffer size typically results in decreased accuracy, potentially due to overfitting to the limited samples. These results show that a larger buffer size may be needed to update the chosen task mask to improve backward transfer effectively.

Table 3. Accuracy and standard deviation of t_c with varied buffer sizes

	CIFAR100	CIFAR100-SC	TinyImageNet	MiniImageNet
LEAF$_{NU}$	78.00 ± 0.48	54.68 ± 0.67	73.60 ± 0.85	65.92 ± 0.82
LEAF$_{20}$	74.36 ± 0.55	50.98 ± 0.68	71.23 ± 0.92	63.74 ± 0.68
LEAF$_{40}$	75.21 ± 0.52	53.79 ± 0.73	74.43 ± 0.51	66.87 ± 0.78
LEAF$_{60}$	77.89 ± 0.67	56.35 ± 0.62	75.65 ± 0.70	66.31 ± 0.58
LEAF$_{80}$	78.34 ± 0.43	56.79 ± 0.45	77.38 ± 0.62	68.72 ± 0.86
LEAF$_{100}$	**79.16 ± 0.31**	**57.20 ± 0.52**	**78.26 ± 0.68**	**69.40 ± 0.74**

5.6 *Training Time and Memory Usage (RQ4)*

We evaluate the efficiency of LEAF in terms of training duration and memory usage against five established CL methods: GPM, TRPG, A-GEM, CUBER, and WSN. These evaluations are conducted using four datasets: TinyImageNet, MiniImageNet, CIFAR100, and CIFAR100-SC, with results visualized in Figure 4. Training time is measured by the total duration after training on all tasks. We record GPU memory consumption after training on a single task to ensure a fair comparison of memory usage, especially between methods that rely on gradient storage versus storing input instances.

CUBER has shown the highest backward transfer among the compared methods but also incurs the longest runtimes across the datasets. The increased run time is primarily due to the computationally intensive process of calculating gradient alignments for each task, which involves handling large gradient vectors. CUBER has the highest memory requirements when using the ResNet-18 backbone on the TinyImageNet and MiniImageNet datasets. In contrast, LEAF typically exhibits lower runtime and memory requirements than CUBER on TinyImageNet and MiniImageNet, primarily due to CUBER's necessity to store extensive gradient vectors. However, when employing the AlexNet model as the backbone for CIFAR100 and CIFAR100-SC, LEAF shows a higher memory requirement due to storing the entire training dataset of the chosen task to facilitate future task mask updates. Despite these variances, LEAF demonstrates similar memory usage to CUBER on TinyImageNet and MiniImageNet but achieves higher backward transfer for the chosen task.

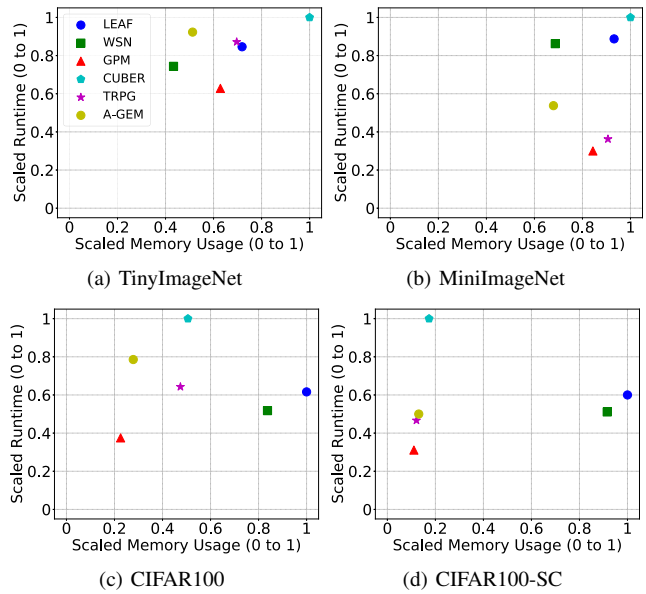

Figure 4. Memory usage is recorded after training on the first task, and runtime is after all tasks are learned. The highest memory usage and runtime recorded are used as the reference (value of 1). Maximum values are recorded on separate axes: CIFAR100 is 112 mins & 361MB, CIFAR100-SC is 90 mins & 347MB, TinyImageNet is 72 mins & 371MB, and MiniImageNet is 160 mins & 310MB.

6 Conclusion and Future Work

In this work, we investigate how to facilitate positive forward and backward transfer of knowledge, using past knowledge to learn new tasks and new knowledge to improve learning on an older task. We proposed and developed a novel CL method, LEAF, which improves backward transfer by introducing a flexible mechanism for updating older knowledge through task mask adaptation. Our experiments demonstrate that LEAF achieves higher BWT_{t_c} in 4 out of 5 benchmark datasets. We also show its effectiveness in a CL fashion as new tasks arrive. We also highlighted scenarios where tasks share similarities and datasets with a higher number of tasks, such as TinyImageNet and MiniImageNet, as LEAF leverages relevant knowledge from new tasks to improve the performance of the chosen task. In future work, we will extend our research to explore strategies that enhance backward transfer under limited memory and simultaneously enable positive backward transfer across multiple tasks.

References

[1] Y. Bulatov. Notmnist dataset. *Google (Books/OCR)*, 2011.

[2] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*, 2019.

[3] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[4] Z. Chen and B. Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

[5] B. Daass, D. Pomorski, and K. Haddadi. Using an adaptive entropy-based threshold for change detection methods – application to fault-tolerant fusion in collaborative mobile robotics. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1173–1178, 2019. doi: 10.1109/CoDIT.2019.8820667.

[6] D. Deng, G. Chen, J. Hao, Q. Wang, and P.-A. Heng. Flattening sharpness for dynamic gradient projection memory benefits continual learning. *Advances in Neural Information Processing Systems*, 34:18710–18721, 2021.

[7] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[8] S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. In *International Conference on Learning Representations*, 2020.

[9] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[10] H. Kang, R. J. L. Mina, S. R. H. Madjid, J. Yoon, M. Hasegawa-Johnson, S. J. Hwang, and C. D. Yoo. Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, pages 10734–10750. PMLR, 2022.

[11] T. Konishi, M. Kurokawa, C. Ono, Z. Ke, G. Kim, and B. Liu. Parameter-Level Soft-Masking for Continual Learning. In *Proc. of ICML*, 2023.

[12] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

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

[15] L. Liebel and M. Körner. Auxiliary tasks in multi-task learning. *arXiv preprint arXiv:1805.06334*, 2018.

[16] S. Lin, L. Yang, D. Fan, and J. Zhang. Beyond not-forgetting: Continual learning with backward knowledge transfer. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[17] S. Lin, L. Yang, D. Fan, and J. Zhang. TRGP: Trust region gradient projection for continual learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=iEvAf8i6JjO.

[18] Y. Luopan, R. Han, Q. Zhang, C. H. Liu, G. Wang, and L. Y. Chen. Fedknow: Federated continual learning with signature task knowledge integration at edge. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 341–354. IEEE, 2023.

[19] W.-C. Ma, S. Wang, M. A. Brubaker, S. Fidler, and R. Urtasun. Find your way by observing the sun and other semantic cues. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2017.

[20] A. Mallya and S. 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.

[21] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[22] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989.

[23] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, pages 11, 16, 2011.

[24] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[25] P. Pan, S. Swaroop, A. Immer, R. Eschenhagen, R. Turner, and M. E. E. Khan. Continual deep learning by functional regularisation of memorable past. *Advances in Neural Information Processing Systems*, 33:4453–4464, 2020.

[26] M. Pietron, D. Żurek, K. Faber, and R. Corizzo. Ada-qpacknet – multi-task forget-free continual learning with quantization driven adaptive pruning. In *26th European Conference on Artificial Intelligence*, 2023.

[27] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902, 2020.

[28] R. Ramesh and P. Chaudhari. Model zoo: A growing brain that learns continually. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=WfvgGBcgbE7.

[29] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[30] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[31] G. Saha, I. Garg, and K. Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=3AOj0RCNC2.

[32] W. Shi, Y. Chen, Z. Zhao, W. Lu, K. Yan, and X. Du. Create and find flatness: Building flat training spaces in advance for continual learning. In *26th European Conference on Artificial Intelligence*, 2023.

[33] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 29, 2016.

[34] W. Wong, Y. S. Koh, and G. Dobbie. Learning after learning: Positive backward transfer in continual learning supplementary material, 2024. Available at https://doi.org/10.5281/zenodo.13357526.

[35] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020.

[36] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[37] A. Xie and C. Finn. Lifelong robotic reinforcement learning by retaining experiences. In *Conference on Lifelong Learning Agents*, pages 838–855. PMLR, 2022.

[38] X. Yan, Y. Luo, and X. Zheng. Weather recognition based on images captured by vision system in vehicle. In *Advances in Neural Networks–ISNN 2009*, pages 390–398, 2009.

[39] J. Yoon, S. Kim, E. Yang, and S. J. Hwang. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1gdj2EKPB.

[40] G. Zeng, Y. Chen, B. Cui, and S. Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019.