Flashback for Continual Learning

Leila Mahmoodi[1,2], Mehrtash Harandi[1,2], Peyman Moghadam[2,3]
[1] Monash University, [2] Data61, CSIRO, [3]Queensland University of Technology
{leila.mahmoodi, mehrtash.harandi}@monash.edu, {leila.mahmoodi, peyman.moghadam}@csiro.au

Abstract

*To strike a delicate balance between model stability and plasticity of continual learning, previous approaches have adopted strategies to guide model updates on new data to preserve old knowledge while implicitly absorbing new information through task objective function (*e.g. *classification loss). However, our goal is to achieve this balance more explicitly, proposing a bi-directional regularization that guides the model in preserving existing knowledge and actively absorbing new knowledge. To address this, we propose the Flashback Learning (FL) algorithm, a two-stage training approach that seamlessly integrates with diverse methods from different continual learning categories. FL creates two knowledge bases; one with high plasticity to control learning and one conservative to prevent forgetting, then it guides the model update using these two knowledge bases. FL significantly improves baseline methods on common image classification datasets such as CIFAR-10, CIFAR-100, and Tiny ImageNet in various settings.*

1. Introduction

Our brain possesses remarkable abilities to acquire new knowledge without forgetting or interfering with the old one. To endow Deep Neural Networks (DNNs) with similar capabilities in computer vision tasks, they must be able to learn new categories, objects, or classes continuously. However, in incremental settings where new categories are introduced one by one, the traditional assumption of having access to the entire dataset upfront is no longer valid, leading to Catastrophic Forgetting (CF) [32] and degradation in DNN performance. Introducing new data causes DNNs to adapt to the new knowledge, often replacing previously learned representations.

Various Continual Learning (CL) methodologies have emerged to address the CF challenge. These methods encompass a diverse range of techniques, such as regularization strategies ([8], [29], [35]) to control the model's evolution, adapting the architecture ([21], [34]) to accommodate new categories, and leveraging memory replay techniques. Memory replay involves using a fraction of previous data ([2], [30]) or generative models to recreate past data ([36], [38], [28]). These methodologies showcase ongoing efforts to equip DNNs with continual learning capabilities.

CL methods aim to effectively transfer old knowledge into a new model while updating it with new task data. Striking a balance between stability (the ability to preserve previously learned information) and plasticity (the capacity to learn further information) poses a central challenge in CL methods [11]. The typical approach in CL involves retaining limited information from previous tasks, such as old samples, a copy of the old model, or its parameters. During the model update on new data, this *information* is utilized in a regularization term to preserve old task knowledge, while the classification loss is responsible for absorbing new task knowledge. Although the old knowledge preservation is deliberately constrained in training, there is a lack of guidance on how to acquire and integrate new task knowledge into the model effectively. This raises the question of how to attain a more balanced approach that explicitly guides both the preservation of old knowledge and the absorption of new knowledge during the learning process.

What if we could utilize same-level *information* to control capacity for new tasks? Intriguingly, despite of protective mechanism within neocortical circuits [39] in our brain, the hippocampus, known for its role in memory consolidation [20], actively updates memories by integrating novel information with existing memory traces [15]. This dynamic process enables enhanced insights and improved adaptation to the current task context [4]. Drawing inspiration from this mechanism, we aim to explore a similar approach to achieve a balanced integration of stability and plasticity when learning new tasks for continual learning.

In this paper, we introduce the Flashback Learning algorithm, a two-step training approach compatible with diverse CL methods. In the initial step, the model is updated on new data using the original CL strategy to preserve old task knowledge. Subsequently, we extract crucial information from this updated model to control model plasticity. In the second step, termed "Flashback", we return to the old model and update the model again, now equipped with the knowledge to guide the preservation of old task knowledge and the acquisition of new task knowledge. We empirically validate the effectiveness of FL algorithm on CIFAR-10, CIFAR-100, and Tiny ImageNet datasets in two CL settings (Task- and Class-Incremental). Our approach outperforms the baselines in both settings across all three datasets. Our contributions are listed below:

- We propose the FL algorithm, which seamlessly integrates with CL methods and leverages their strategy to update the model on new data and extract new knowledge. Unlike traditional CL methods, FL guides the model bidirectionally, combining old and new knowledge in its second stage. This balanced approach significantly enhances CL model performance.
- FL is compatible with various CL methods, including replay-based, distillation, and parameter-regularization techniques. Combined with FL, these methods show substantial accuracy improvement in both Task-Incremental (TI) and Class-Incremental (CI) settings.
- Extensive experiments validate the effectiveness of FL, showcasing its advantage over baselines within the same training epochs. These empirical findings show FL's potential to advance CL models' performance and open new possibilities.

2. Notations

In continual learning, a model is trained sequentially on tasks $\mathcal{T}=\{t\}_{t=1}^{T}$. Within each task, t, inputs $\mathbf{x}$, and their corresponding ground truth labels $\mathbf{y}$ are drawn from data distribution $D_t = (\mathcal{X}_t, \mathcal{Y}_t)$. Here, $\mathcal{X}_t$ and $\mathcal{Y}_t$ represent the sets of inputs and outputs specific to task t. Task t data originates from a distinct set of classes C_t in the considered settings, with no overlap between classes across different tasks. In TI scenario, task identifier t is available during evaluation, allowing the model to concentrate decision boundaries within individual tasks. The CL model needs decision-making on all seen classes during inference in the more demanding CI setting [42]. While training on task t, the model is denoted as $f_t(.;\boldsymbol{\theta})$, and at the conclusion of the task, the updated model is represented as $f_t(.;\boldsymbol{\theta}^*)$; where $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^*$ are respectively model parameters during and after training. The model is decomposed into two parts: $f_t(.;\boldsymbol{\theta}) = g \circ h(.;\boldsymbol{\theta})$, where feature extractor $h : \mathcal{X}_t \rightarrow \mathbb{R}^d$ maps input image $\mathbf{x}$ to a low-dimensional feature vector $\mathbf{h}$. The linear classifier $g : \mathbb{R}^d \rightarrow \mathbb{R}^c$ maps the latent feature $\mathbf{h}$ to classification logits $\mathbf{z}$, where c denotes the total number of classes.

3. Flashback Learning

Flashback Learning emphasizes the simultaneous control of knowledge absorption and retention. Initially, the FL algorithm updates the old model $f_s(.;\boldsymbol{\theta}_s)$ with new data to obtain the primary model $f_p(.;\boldsymbol{\theta}_p)$. In this initial phase, FL has access to some knowledge from the previous task; this knowledge is determined based on the CL method, to which FL is added. Having observed new data for some epochs, $f_p(.;\boldsymbol{\theta}_p)$ acquired some knowledge of the new task. By the end of the first step, FL extracts new knowledge using $f_p(.;\boldsymbol{\theta}_p)$; then it returns to the old model and updates it within the final step; while benefiting from dual guidance by previous and new knowledge jointly.

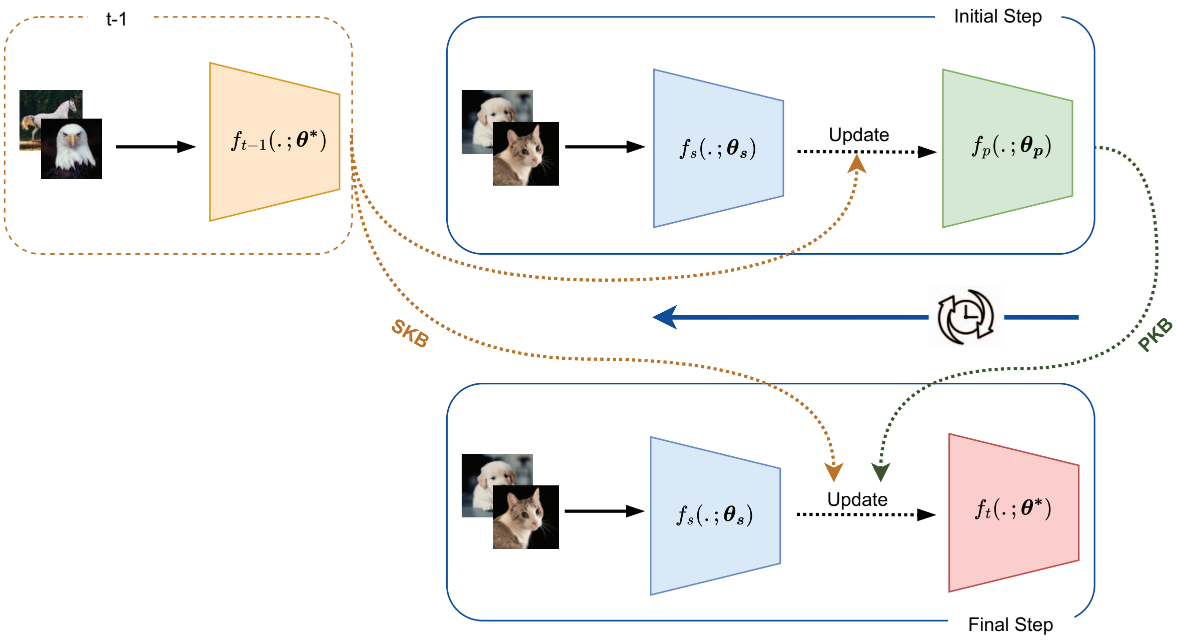
FL algorithm is compatible with various CL methods, including replay-based, distillation-based, and parameter-regularization methods. It updates the model from the previous task to the new one in two steps. In the first stage, it takes a copy of the old model (see $f_s(.;\boldsymbol{\theta}_s)$ in Figure 1) and follows the original CL strategy to update the model for limited epochs to obtain a primary model (see $f_p(.;\boldsymbol{\theta}_p)$ in Figure 1). CL methods offer access to different levels of old information to control knowledge retention. We refer to this information as the Stable Knowledge Base (SKB). For instance, replay methods provide access to a subset of old samples and responses by which model updates can be regularized in the new task. After the initial step, we extract similar information using primary model $f_p(.;\boldsymbol{\theta}_p) = g_p \circ h_p(.;\boldsymbol{\theta}_p)$ and store it in a separate knowledge base called the Plastic Knowledge Base (PKB).

In the final stage, our algorithm returns to $f_s(.;\boldsymbol{\theta}_s)$ (see *Final Step* Figure 1) and continues updating the model with new data by a bidirectional regularization. This means that, in addition to the classification loss, we utilize the SKB to control model stability and the PKB to guide model plasticity. This comprehensive approach ensures a more balanced and effective training process that actively considers knowledge retention and absorption. As a result, the FL algorithm significantly enhances the performance of the CL model in achieving a better trade-off between preserving old knowledge and acquiring new knowledge.

Section 3.1 provides an explanation of SKB within the FL framework and defines SKB for different categories of CL methods. In Section 3.2, we present the PKB and show how it is constructed to align with SKB. We detail how the FL algorithm effectively utilizes these two knowledge bases to regulate the model updates in its training objective in Section 3.3.

3.1. Stable Knowledge Base

SKB serves as a supportive agent, helping the model remember and preserve old knowledge after each update. Providing information from previous tasks, SKB controls forgetting and retains valuable knowledge from past experiences. Here, we define SKB, denoted as S in FL setting. The specific content of existing knowledge varies depending on the strategy employed by the CL method to preserve



t-1
$f_{t-1}(.;\boldsymbol{\theta^*})$
Initial Step
$f_s(.;\boldsymbol{\theta_s})$
Update
$f_p(.;\boldsymbol{\theta_p})$
SKB
PKB
$f_s(.;\boldsymbol{\theta_s})$
Update
$f_t(.;\boldsymbol{\theta^*})$
Final Step

Figure 1. Flashback Learning Algorithm; **Initial Step**- starts from old model (*i.e.* $f_s(.;\boldsymbol{\theta}_s) = f_{t-1}(.;\boldsymbol{\theta}^*)$), and updates the old model with new data for limited epochs while constrained by SKB to obtain primary model $f_p(.;\boldsymbol{\theta}_p)$. **Final Step**-flashback to $f_s(.;\boldsymbol{\theta}_s)$ and update again while constrained by SKB and PKB simultaneously, results in final new model $f_t(.;\boldsymbol{\theta}^*)$

knowledge. By categorizing CL methods into three groups, we can define SKB and identify the accessible information S within each category.

3.1.1 Distillation

In distillation methods, knowledge is transferred from the old to the new model to preserve previously learned information; and the loss function for updating the model on the current task contains a distillation term. This loss component requires access to the old model, as it involves distilling the knowledge from the old model into the new one. In practical terms, this means that a copy of the old model is used to constrain the representations of the new model at different stages, such as after the classifier [25], in intermediate stages [12], or after the feature extractor [18], to prevent their deviation from the old ones. In this category, SKB comprises a copy of the entire old model $f_s(.;\boldsymbol{\theta}_s)$ or the old feature extractor $h_s(.)$, which serves as the stable knowledge S;

$$S \triangleq \{f_s(.;\boldsymbol{\theta}_s), h_s(.)\}. \tag{1}$$

3.1.2 Memory Replay

In replay-based methods, the available knowledge S in SKB includes a small subset of old samples $\mathbf{x}$ and their corresponding ground-truth labels $\mathbf{y}$. These samples are stored in a memory buffer $\mathcal{M}_{t-1}$ and interleaved with the current data for joint training. In some replay methods, additional information is also kept alongside the labels. For example, in DER/DER++ [5], the old model's logits $\mathbf{z}_s = f_s(\mathbf{x};\boldsymbol{\theta}_s)$ are stored alongside the images and labels. Similarly, in [19], lower-dimensional feature embeddings $\mathbf{h}_s = h_s(\mathbf{x})$ are retained with the labels in the memory buffer. Therefore, in replay-based methods, S is retrieved from the following SKB set;

$$\begin{aligned} S \triangleq \Big\{&\mathbf{x},\, \mathbf{h}_s = h_s(\mathbf{x}),\, \mathbf{z}_s = f_s(\mathbf{x};\boldsymbol{\theta}_s),\, \mathbf{y} \mid \\ &(\mathbf{x},\mathbf{y}) \sim \mathcal{M}_{t-1}\Big\}. \end{aligned} \tag{2}$$

in which $(\mathbf{x},\mathbf{y})$ is sampled from memory buffer $\mathcal{M}_{t-1}$. Based on replay strategy, $\mathbf{h}_s$ or $\mathbf{z}_s$ is the old model response to $\mathbf{x}$, kept in SKB.

3.1.3 Parameter Regularization

Critical parameters from the previous model regularize the updating process for the new task in this category. Therefore, parameter regularization methods have access to a copy of the old model's weights and a measure of their importance. This importance measure is computed using the Fisher matrix, as employed by the EWC method [23]. The online EWC approach [35] updates the overall Fisher matrix $\bar{\mathbf{F}}_t$ recursively, incorporating the Fisher matrix $\mathbf{F}_t$ from each task as follows;

$$\bar{\mathbf{F}}_t = \gamma\bar{\mathbf{F}}_{t-1} + \mathbf{F}_t \mid t = 1 \ldots \mathcal{T},\, \bar{\mathbf{F}}_1 = \mathbf{F}_1, \tag{3}$$

Algorithm 1: Flashback Learning Algorithm

```
Input: Old model $f_{t-1}(.;\boldsymbol{\theta}^*)$, New data
       $(\mathbf{x},\mathbf{y}) \in (\mathcal{X}_t,\mathcal{Y}_t)$ , Available SKB Eq. (1)-(4)
Output: New model $f_t(.;\boldsymbol{\theta}^*)$, Updated SKB
$f_s(.;\boldsymbol{\theta}_s) = f_{t-1}(.;\boldsymbol{\theta}^*)$
$f_t(.;\boldsymbol{\theta}) = f_{t-1}(.;\boldsymbol{\theta}^*)$
for epoch = 1, $\cdots$ , $E_1$ do
    for batch = 1, $\cdots$ , $\mathcal{B}$ do
        Generate model response $f_t(\mathbf{x};\boldsymbol{\theta})$
        // Classification loss
        $\mathcal{L}_c(\boldsymbol{\theta}) = \mathcal{L}_c(f_t(\mathbf{x};\boldsymbol{\theta}),\mathbf{y})$
        // Use SKB Eq. 1-4, calculate $\mathcal{L}_s(\boldsymbol{\theta})$
        $\mathcal{L}_s(\boldsymbol{\theta}, S)$ // Eq. 10-11
        Let $\mathcal{L}_1(\boldsymbol{\theta}) = \mathcal{L}_c(\boldsymbol{\theta}) + \alpha_s\mathcal{L}_s(\boldsymbol{\theta})$
        Update $\boldsymbol{\theta}$// Descending gradient $\mathcal{L}_1(\boldsymbol{\theta})$
    end
end
$f_p(.;\boldsymbol{\theta}_p) = f_t(.;\boldsymbol{\theta})$ // primary model
$P \in PKB$ // Extract PKB Eq. 5-7
$f_t(.;\boldsymbol{\theta}) = f_s(.;\boldsymbol{\theta}_s)$ // Flashback
for epoch = 1, $\cdots$ , $E_2$ do
    for batch = 1, $\cdots$ , $\mathcal{B}$ do
        Generate model response $f_t(\mathbf{x};\boldsymbol{\theta})$
        // Classification loss
        $\mathcal{L}_c(\boldsymbol{\theta}) = \mathcal{L}_c(f_t(\mathbf{x};\boldsymbol{\theta}),\mathbf{y})$
        // Stability loss, Eq. 10-11
        Use SKB to calculate $\mathcal{L}_s(\boldsymbol{\theta})$
        // Plasticity loss, Eq. 14-15
        Use PKB to calculate $\mathcal{L}_p(\boldsymbol{\theta})$
        Let $\mathcal{L}_2(\boldsymbol{\theta}) = \mathcal{L}_c(\boldsymbol{\theta}) + \alpha_s\mathcal{L}_s(\boldsymbol{\theta}) + \alpha_p\mathcal{L}_p(\boldsymbol{\theta})$
        Update $\boldsymbol{\theta}$ // Descending gradient $\mathcal{L}_2(\boldsymbol{\theta})$
    end
end
Update SKB for the next task
```

where $\mathbf{F}_1$ is the Fisher matrix calculated for the first task and $\gamma < 1$ is the hyperparameter to adjust the contribution of the previous Fisher approximation $\bar{\mathbf{F}}_{t-1}$. This recursive update process prevents the computational complexity from growing linearly with the number of tasks. The SKB in parameter-regularization methods includes empirical Fisher matrix $\bar{\mathbf{F}}_t$ and the old parameters.

$$S \triangleq \{\boldsymbol{\theta}_s = \boldsymbol{\theta}^*,\ \mathbf{F_s} = \bar{\mathbf{F}}_t\}. \tag{4}$$

Let $\boldsymbol{\theta}^*$ be the old model parameters and $\bar{\mathbf{F}}_t$ is the average Fisher matrix calculated by (3).

3.2. Plastic Knowledge Base

In the FL algorithm, we utilize stable knowledge S from SKB and apply the same updating strategy as the original CL method to update the old model on the new task during the initial training (see *Initial Step* in Figure 1). The first step's output is primary model $f_p(,;\boldsymbol{\theta}_p)$, which allows extracting same-level information to SKB (*e.g.* in replay method we have $\mathbf{x}$, sampled from the memory buffer, and its corresponding old response; so we equally generate the primary model response to $\mathbf{x}$.) We refer to this extracted information as plastic knowledge P, which is kept in the Plastic Knowledge Base (PKB) to adjust the model's plasticity in FL final training. PKB acts as an informed agent; having observed new data, it guides FL to integrate the new knowledge into the model effectively. For distillation methods, same as SKB (1), PKB contains a copy of the entire primary model $f_p(.;\boldsymbol{\theta}_p)$ or the feature extractor $h_p(.)$;

$$P \triangleq \{f_p(.;\boldsymbol{\theta}_p), h_p(.)\}. \tag{5}$$

Considering stable knowledge S in replay-based SKB (2), we generate primary model response for buffer samples as plastic knowledge P;

$$P \triangleq \Big\{\mathbf{h}_p = h_p(\mathbf{x}), \mathbf{z}_p = f_p(\mathbf{x};\boldsymbol{\theta}_p) \mid \mathbf{x} \sim \mathcal{M}_{t-1}\Big\}; \tag{6}$$

here, $\mathbf{x}$ is sampled from the memory buffer $\mathcal{M}_{t-1}$, based on replay strategy $\mathbf{h}_p$ or $\mathbf{z}_p$ are primary model responses, kept in PKB. If the CL strategy involves parameter regularization by stable knowledge (4), we compute an additional Fisher matrix, $\mathbf{F_p}$, to assess the importance of parameters in the primary model $f_p(,;\boldsymbol{\theta}_p)$. A copy of the primary model's weights and the corresponding Fisher matrix is then added to the PKB as;

$$P \triangleq \{\boldsymbol{\theta}_p, \mathbf{F_p}\}, \tag{7}$$

to ensure consistency between the two knowledge bases. In Eq. (7), $\boldsymbol{\theta}_p$ is a copy of primary model weights, and $\mathbf{F_p}$ is its Fisher matrix.

3.3. Flashback Learning Steps

3.3.1 Initial Step

In this step, we start by creating a copy of the old model as $f_s(,;\boldsymbol{\theta}_s)$. We then update the model using the same objective function as the original CL method, but only for a limited number of epochs E_1. The loss function of the original CL method is split into two components;

$$\mathcal{L}_1(\boldsymbol{\theta}) = \alpha_s\mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_c(\boldsymbol{\theta}), \tag{8}$$

$\mathcal{L}_s(\boldsymbol{\theta})$ is the stability loss, taking stable knowledge S from SKB to regularize the training process, and α_s is its hyperparameter. The second component of the loss function $\mathcal{L}_c(\boldsymbol{\theta})$ is the summation of the remaining terms, including the classification loss. The second component optimizes the model's performance on the new task. The stability loss $\mathcal{L}_s(\boldsymbol{\theta})$ is formulated particularly for each CL method. For replay-based methods like DER [5], $\mathcal{L}_s(\boldsymbol{\theta})$ is computed as the mean squared error (MSE) loss between the new model's response and the old logits from replay SKB (2);

$$\mathcal{L}_s(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathbf{z}_s)\sim SKB}\left[\left\|f_t(\mathbf{x};\boldsymbol{\theta}) - \mathbf{z}_s\right\|_2^2\right]. \tag{9}$$

Here, stability loss directs the model to retain consistency with the stable knowledge in SKB; $\mathbf{x}$ is sampled from the memory buffer, and $\mathbf{z}_s$ is its corresponding old logit kept in SKB.

Stability loss is designed for distillation-based methods to constrain the new model's representation at a particular stage. In LUCIR [18], the stability loss controls the final feature embeddings of the model. LUCIR stability loss is formulated by cosine embedding loss, which employs a copy of old feature extractor $h_s(.)$ kept in SKB (1). In practice, $\mathcal{L}_s(\boldsymbol{\theta})$ encourages the new and old embeddings to remain adjacent throughout the update process;

$$\mathcal{L}_s(\boldsymbol{\theta}) = \left(1 - \langle \bar{h}(\mathbf{x}) - \bar{h}_s(\mathbf{x})\rangle\right). \tag{10}$$

Let $\bar{h} = h/\|h\|_2$ represent the ℓ_2-normalized vector of feature embeddings and $\langle\cdot,\cdot\rangle$ is the inner product between two vectors. $\bar{h}(\mathbf{x})$ represents the normalized feature embedding from current model and $\bar{h}_s(\mathbf{x})$ is the normalized feature embedding generated by old model.

Stability loss for parameter regularization takes old parameters from SKB (4) to control new parameters' updates directly. The regularization term is weighted by the empirical Fisher matrix $\mathbf{F_s}$ stored as stable knowledge (4);

$$\mathcal{L}_s(\boldsymbol{\theta}) = \left\|\boldsymbol{\theta} - \boldsymbol{\theta}_s\right\|_{\mathbf{F_s}}^2, \tag{11}$$

to force crucial parameters remain close to their old values. In Eq. (11), $\|\mathbf{v}\|_{\mathbf{F}}^2 = \mathbf{v}^\top\mathbf{F}\mathbf{v}$.

In FL's first stage, primary model $f_p(.;\boldsymbol{\theta}_p)$ is obtained by descending gradient on $\mathcal{L}_1(\boldsymbol{\theta})$ for the limited number of epochs E_1, stability loss $\mathcal{L}_s(\boldsymbol{\theta})$ in $\mathcal{L}_1(\boldsymbol{\theta})$ is formulated differently based on available stable knowledge and CL strategy in knowledge retention. We derive consistent information with SKB after obtaining $f_p(.;\boldsymbol{\theta}_p)$ and store it in the PKB to serve the requirement of the final step in the FL algorithm.

3.3.2 Final Step

In this stage, we reinitialize the model to $f_s(.;\boldsymbol{\theta}_s)$ and proceed with the model update on new data for an adequate number of epochs E_2 by the modified objective function

$$\mathcal{L}_2(\boldsymbol{\theta}) = \mathcal{L}_c(\boldsymbol{\theta}) + \alpha_s\mathcal{L}_s(\boldsymbol{\theta}) + \alpha_p\mathcal{L}_p(\boldsymbol{\theta}). \tag{12}$$

The first two terms of $\mathcal{L}_c(\boldsymbol{\theta})$ and $\mathcal{L}_s(\boldsymbol{\theta})$ are formulated analogous to the first step of the FL algorithm (see Eq. (8)-(11)). The additional term $\mathcal{L}_p(\boldsymbol{\theta})$ is introduced to control the model's plasticity, and α_p is its hyperparameter. As discussed in Section 3.2, PKB is built from the extracted information of primary model $f_p(.;\boldsymbol{\theta}_p)$, following a similar format as SKB. Consequently, $\mathcal{L}_p(\boldsymbol{\theta})$ is defined the same as stability loss but incorporating plastic knowledge P from PKB.

Considering MSE stability loss $\mathcal{L}_s(\boldsymbol{\theta})$ for replay methods in Eq. (9), plasticity loss is defined as;

$$\mathcal{L}_p(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{M}_{t-1}\,\mathbf{z}_p\sim PKB}\left[\left\|f_t(\mathbf{x};\boldsymbol{\theta}) - \mathbf{z}_p\right\|_2^2\right]; \tag{13}$$

$\mathbf{x}$ is sampled from the memory buffer $\mathcal{M}_{t-1}$, and $\mathbf{z}_p$ is its corresponding logit generated by the primary model and kept in Eq. (6).

Distillation-based methods with stability loss (10), formulate $\mathcal{L}_p(\boldsymbol{\theta})$ as cosine embedding loss;

$$\mathcal{L}_p(\boldsymbol{\theta}) = \left(1 - \langle \bar{h}(\mathbf{x}) - \bar{h}_p(\mathbf{x})\rangle\right), \tag{14}$$

in which $\bar{h}(\mathbf{x})$ and $\bar{h}_p(\mathbf{x})$ are respectively normalized feature embedding from new and primary models.

Plasticity loss $\mathcal{L}_p(\boldsymbol{\theta})$ for parameter regularization methods follows

$$\mathcal{L}_p(\boldsymbol{\theta}) = \left\|\boldsymbol{\theta} - \boldsymbol{\theta}_p\right\|_{\mathbf{F_p}}^2. \tag{15}$$

to go with the stability loss $\mathcal{L}_s(\boldsymbol{\theta})$ in Eq. (11); $\boldsymbol{\theta}_p$ shows the primary model's parameters and $\mathbf{F_p}$ is its Fisher matrix.

In the final FL step, we again start from the old model $f_s(.;\boldsymbol{\theta}_s)$ and update it on new data for the adequate number of epochs (denoted as E_2). Stability and plasticity loss components regulate the model updates in this step. The first term aims to preserve the stable knowledge obtained from the previous tasks, while the plasticity one counteracts the first one, ensuring that the model remains close to the plastic knowledge to encourage knowledge acquisition from the new task(see Algorithm 1). By incorporating these two regularizers, we achieve a better balance of stability and plasticity in the model's sequential learning process. We expect that this modification in the loss function will result in improved accuracy for the CL model, reflecting FL's ability to handle incremental learning scenarios more effectively.

3.4. Limitation

In FL, initially, we update the model for a limited number of epochs (E_1). In our experiments, we determine E_1 such that the total number of training epochs of FL matches that of baseline techniques (*i.e.*, $E_1 + E_2$ is equal to the number of training epochs for the baseline CL techniques). This consideration ensures the FL algorithm maintains a comparable training duration to the baseline CL methods.

Determining the appropriate value for E_1 remains a challenge in our method, as it depends on the specific CL setting and dataset characteristics. Additionally, in Eq. (12), the contributions of stability and plasticity losses are scaled by hyperparameters α_s and α_p, respectively. Tuning these hyperparameters is another challenge that influences the FL algorithm's performance. Despite these challenges, the potential applicability of FL encourages us to address these limitations in future research.

4. Related Works

4.1. Memory Replay

Replay-based methods [31], [10], [6] utilize a small memory buffer of samples and their corresponding labels from previous tasks. These stored examples are combined with new data during training on a new task. Other than images and labels, some methods store lower-dimensional features [7], [19], or logits generated by the old model in the memory buffer for knowledge retention. Different strategies have been proposed to make use of the memory buffer to preserve old information; DER/DER++ [5] and X-DER [3] augment the cross-entropy loss with a logit distillation component during the replay process. GEM [27] and A-GEM [9] impose constraints on new task optimization by memory samples. Some methods employ generative models to generate old samples [26], [40], or representations [38] for replay. Unlike traditional replay methods that only rely on the memory buffer's old information to preserve stable knowledge, the FL algorithm generates and accumulates counterpart information to adjust plasticity concurrently.

4.2. Parameter Regularization

Parameter regularization methods aim to understand how modifications to model weights affect task losses; subsequently, they employ strategies to restrict updates for critical old parameters and preserve old knowledge. One notable approach in this field is EWC [23], which introduces the Fisher Information Matrix to evaluate the importance of weights. Other methods, such as SI [41] and MAS [1], utilize estimated path integrals and gradient magnitudes to regulate weight changes. RWalk [8] combines the Fisher information matrix and online path integral for approximating parameter importance. Online EWC [35] presents a solution for efficient computation and use of the average Fisher matrix. In this paper, an additional Fisher matrix is computed after the initial training to emphasize the importance of crucial new parameters. Then in the final training, the interplay between the two stability and plasticity regularization terms improves model performance as it enables the preservation of relevant knowledge while allowing for the necessary capacity for the new task.

4.3. Distillation

Knowledge distillation methods, known for transferring knowledge from a larger teacher model to a smaller student one, have been adapted to preserve old knowledge in CL methods. Knowledge distillation methods focus on regulating intermediate outputs while the model gets updated on the new task. For instance, [25] proposed regularization on logits between the new model and the old checkpoint. Similarly, [37] and [33] perform knowledge distillation on final feature maps before the classification layer. PODNet [12] extended knowledge distillation to intermediate feature maps, and AFC [22] incorporated gradient information to weigh the feature distillation loss to preserve essential features for old tasks. Recent advancements in distillation approaches have focused on achieving a better balance between stability and plasticity by incorporating techniques such as projecting new features onto the old ones within a contrastive learning framework [13, 14] or utilizing a learnable linear transformation [16]. When integrating FL into distillation methods, we observe improved accuracy in both CI and TI settings without the need for additional contrastive learning or learning a transformation.

5. Experiments

To assess the effectiveness of our proposed approach, we conducted experiments on image classification datasets commonly used in CL literature;

- **Split CIFAR-10** [41],[30]: CIFAR-10 dataset [24] is split to 5 sequential tasks; each includes 2 classes with 5000 and 1000 32×32 colored images, respectively for training and testing.
- **Split CIFAR-100** [10],[41]: CIFAR-100 dataset [24] is split to 10 sequential tasks; each includes 10 classes. Each class comprises 500 and 100, train and test 32×32 colored images.
- **Split Tiny ImageNet** [5]: Tiny ImageNet dataset contains 100000 train and 10000 test downsized 64×64 colored images across 200 classes and is split into 10 tasks of 20 classes.

5.1. Implementation Details

We adopted the ResNet18 [17] architecture as the base model for our experiments as commonly used in [3], [5] on the image classification datasets. The models were trained from scratch without any pre-training. Following the approach of previous studies [3], [5], [30], we employed a multi-epoch setup, allowing for multiple passes on the training data for each task. Specifically, for the Split CIFAR-10 and CIFAR-100 datasets, we trained the models for 60 epochs per task, while for the Split Tiny ImageNet dataset,

Table 1. FL algorithm in Class-Incremental (CI) setting, combined with different CL methods. Average Accuracy (↑) for standard CL benchmarks is reported. For replay methods (**iCaRL** and **X-DER**), the buffer size is respectively set to 500, 2000, and 5120, for Split CIFAR-10, Split CIFAR-100, and Split Tiny ImageNet. (*) denotes results from [3] and [5] using the same implementation.

	Split CIFAR-10	**Split CIFAR-100**	**Split Tiny ImageNet**
	5 tasks	**10 tasks**	**10 tasks**
New classes per task	2	10	20
iCaRL [30]	56.56	49.82*	27.32
iCaRL+FL (ours)	**64.77**	**53.16**	**31.18**
LwF.MC [30]	43.14	16.22*	24.18
LwF-MC+FL (ours)	**45.88**	**21.13**	**25.58**
X-DER [3]	56.85	59.14*	40.72
X-DER+FL (ours)	**64.31**	58.02	**41.60**

Table 2. FL algorithm in Task-Incremental (TI) setting, combined with different CL methods. Average Accuracy (↑) for standard CL benchmarks is reported. For replay methods (**iCaRL** and **X-DER**), the buffer size is respectively set to 500, 2000, and 5120, for Split CIFAR-10, Split CIFAR-100, and Split Tiny ImageNet. (*) denotes results from [5] and [5] using the same implementation.

	Split CIFAR-10	**Split CIFAR-100**	**Split Tiny ImageNet**
	5 tasks	**10 tasks**	**10 tasks**
New classes per task	2	10	20
iCaRL [30]	88.7	84.16	68.28
iCaRL+FL (ours)	**91.03**	**86.43**	**69.8**
LwF.MC [30]	93.28	61.40	65.45
LwF.MC+FL (ours)	**96.79**	**64.59**	**67.21**
oEWC [35]	68.29*	59.34	19.20*
EWCo+FL (ours)	**70.85**	**62.35**	**21.01**
X-DER [3]	92.89	88.89	75.16
X-DER+FL (ours)	**94.85**	**89.49**	**76.53**

we trained for 80 epochs. We use Stochastic Gradient Descent (SGD) with a predefined schedule for decreasing the learning rate at specific epochs, the same epochs and weight decay rate for the original **CL** and **CL+FL** methods.

It is important to note that comparing different baselines with intricate yet meaningful differences in experimental settings can be challenging. We employed available implementations of all baselines within a unified experimental environment to address this issue, using the provided codebase [1]. This approach ensures consistent evaluations across the methods. We followed the guidelines and recommendations reported in [3] for hyperparameter selection in the CL methods. Subsequently, we adjusted the hyperparameters related to PKB information for each CL method.

5.2. Metrics and Baselines

To evaluate the performance of Flashback Learning, we examine the Average Accuracy metric before and after integrating Flashback into the CL methods. We report the results in CI and TI settings. After training the model on all $\mathcal{T}$ tasks, we denote the accuracy of the model on the test dataset of task i as A_i and define the Average Accuracy as

$$\bar{A} = \frac{1}{\mathcal{T}} \sum_{i=1}^{\mathcal{T}} A_i. \tag{16}$$

Average Accuracy is a concise and immediate measure that enables straightforward comparisons among different baselines. In addition, we use specific CL metrics such as Forgetting [8] to measure the extent of forgetting previous tasks. We analyze Backward Transfer (BWT) [27] to assess the impact of learning task t on previous tasks, where positive BWT signifies performance improvement on preceding task k. Moreover, we examine Forward Transfer (FWT) [27] to quantify the influence of learning task t on the performance of future tasks. We use the following CL methods and add our FL algorithm:

- **Incremental Classifier and Representation Learning (iCaRL)** [30] uses a memory buffer populated by a herding strategy and distillation to preserve knowledge. ICaRL combines samples stored in the memory with new ones for joint training and distills knowledge from the old model by focusing on logits.
- **Learning without Forgetting (LwF)** [25] is a distillation approach that employs the old model as a teacher

[1]https://github.com/aimagelab/mammoth.

during the current task. In particular, we add FL to the **LwF.MC**, an adaptation of LwF designed in [30].

- **Online Elastic Weight Consolidation (o-EWC)** [35] calculates the Fisher matrix online to assess the importance of parameters in previous tasks and constrain critical ones to preserve old knowledge when updating the model on the new task.

- **Extended Dark Experience Replay (X-DER)** [3] keeps old samples, corresponding labels, and old model logits in the memory buffer for upcoming tasks. This method considers memory updates and preparation for future classes in its objective function.

5.3. Results

For our experimental analysis, we chose at least one method from the CL category to integrate the FL algorithm. While these selected methods may not necessarily represent state-of-the-art accuracy, they allow us to examine the impact of FL on model performance across different CL approaches. We respectively selected iCaRL, LwF, and oEWC as representatives of replay, distillation, and parameter regularization categories. Additionally, we included the X-DER method in our selection as the current state-of-the-art method in the replay category.

We present the average accuracy of CL methods before and after integrating the FL algorithm for both the CI and TI settings (see Table 1 and Table 2, respectively). Notably, CL methods that combine memory replay and distillation to prioritize stability exhibit a more significant improvement with the addition of FL. Our initial observation of this performance boost was in the case of iCaRL, and then we investigated X-DER from the memory replay and distillation category. Experimental results for X-DER on all benchmarks with FL learning show a significant improvement in average accuracy except for Split-CIFAR100 in the CI setting (Table 1). We acknowledge that the original source (X-DER) [3] reports an accuracy of $59.15\,\%$ and we report the same in Table 1, but we independently reproduced an average accuracy of $57.57\,\%$ using their code base.

We analyzed the FWT metric for LwF.MC, X-DER, and oEWC on Split CIFAR-10 and Split CIFAR-100 in TI setting. Table 4 shows that FWT consistently improves across all rows with FL integrated, inferring that learning a new task positively impacts the model's capacity for handling upcoming tasks. In our observations (see Table 3), we noticed that for CL methods like X-DER and iCARL, which utilize distillation to retain old knowledge from memory samples, the FL algorithm's plasticity guidance can counteract knowledge preservation, resulting in smaller BWT and larger Forgetting. Thus, careful selection of hyperparameters controlling stability and plasticity in the final step is crucial for FL's success. On the other hand, for methods like oEWC and LwF.MC, incorporating FL not only improves Backward Transfer and reduces Forgetting but also enhances their performance in acquiring knowledge more efficiently (see Table 4).

Table 3. Backward Transfer (↑) and Forgetting (↓). Results for Split CIFAR-10 in Task-Incremental setting before and after Flashback Learning.

Method	Backward Transfer (↑)	Forgetting (↓)
LwF.MC	-2.31	2.31
LwF.MC+FL (ours)	-2.2	2.2
X-DER	-1.25	1.4
X-DER+FL (ours)	-2.28	2.3
oEWC	-19.93	21.9
oEWC+FL (ours)	-13.01	13.47
iCaRL	0.16	0.11
iCaRL+FL (ours)	-2.03	2.03

Table 4. Forward Transfer (↑) Results in Task-Incremental setting before and after Flashback Learning.

Method	Split CIFAR-10	Split CIFAR-100
LwF.MC	0.85	-0.43
LwF.MC+FL (ours)	2.3	1.83
X-DER	-2.1	1.63
X-DER+FL (ours)	1.68	2.93
oEWC	2.96	-0.16
oEWC+FL (ours)	3.27	1.23

Acknowledgments

P.M. acknowledges co-funding of the project by the CSIRO's Machine Learning and Artificial Intelligence Future Science Platform (CSIRO's MLAI FSP). L.M. also acknowledges funding from the CSIRO's Data61 Postgraduate Scholarship.

6. Conclusion

FL algorithm introduces an innovative approach in CL setting to control model plasticity and stability explicitly by two regularization terms next to the task objective function. This stands in contrast to the common practice in literature, where knowledge retention is constrained by stable knowledge from one side, and knowledge absorption is left to the task objective function (*e.g.* classification loss) without active control over plasticity. Our experimental analysis demonstrates that this bidirectional regularization in FL leads to improved performance and a more balanced trade-off between preserving old knowledge and learning new knowledge. As the FL algorithm exhibits potential applicability to various CL categories, our future work will focus on further exploring and refining this mechanism.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory Aware Synapses: Learning what (not) to forget. In *Eur. Conf. Comput. Vis.*, 2018.

[2] Lorenzo Bonicelli, Matteo Boschini, Angelo Porrello, Concetto Spampinato, and Simone Calderara. On the effectiveness of lipschitz-driven rehearsal in continual learning. *Adv. Neural Inform. Process. Syst.*, 35:31886–31901, 2022.

[3] Matteo Boschini, Lorenzo Bonicelli, Pietro Buzzega, Angelo Porrello, and Simone Calderara. Class-incremental continual learning into the extended der-verse. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(5):5497–5512, 2022.

[4] Donna J Bridge and Joel L Voss. Hippocampal binding of novel information with dominant memory traces can support both memory stability and change. *Journal of Neuroscience*, 34(6):2203–2213, 2014.

[5] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Adv. Neural Inform. Process. Syst.*, 33:15920–15930, 2020.

[6] Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. New insights on reducing abrupt representation change in online continual learning. In *Int. Conf. Learn. Represent.*, 2022.

[7] Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with adaptive quantization modules. In *Int. Conf. Mach. Learn.*, pages 1240–1250. PMLR, 2020.

[8] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Eur. Conf. Comput. Vis.*, pages 532–547, 2018.

[9] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-GEM. In *Int. Conf. Learn. Represent.*, 2019.

[10] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *Proc. of Machine Learn. Research*, 2019.

[11] 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 Trans. Pattern Anal. Mach. Intell.*, 44(7):3366–3385, 2021.

[12] Arthur Douillard, Matthieu Cord, Charles Ollion, and Thomas Robert. PODNet: Pooled Outputs Distillation for Small-Tasks Incremental Learning. In *Eur. Conf. Comput. Vis.*, 2020.

[13] Enrico Fini, Victor G Turrisi Da Costa, Xavier Alameda-Pineda, Elisa Ricci, Karteek Alahari, and Julien Mairal. Self-supervised models are continual learners. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 9621–9630, 2022.

[14] Alex Gomez-Villa, Bartlomiej Twardowski, Lu Yu, Andrew D Bagdanov, and Joost van de Weijer. Continually learning self-supervised representations with projected functional regularization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3867–3877, 2022.

[15] Oscar C González, Yury Sokolov, Giri P Krishnan, Jean Erik Delanois, and Maxim Bazhenov. Can sleep protect memories from catastrophic forgetting? *eLife*, 9:e51005, 2020.

[16] Qiao Gu, Dongsub Shim, and Florian Shkurti. Preserving linear separability in continual learning by backward feature projection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 24286–24295, 2023.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[18] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a Unified Classifier Incrementally via Rebalancing. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[19] Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *Eur. Conf. Comput. Vis.*, pages 699–715. Springer, 2020.

[20] Daoyun Ji and Matthew A Wilson. Coordinated memory replay in the visual cortex and hippocampus during sleep. *Nature neuroscience*, 10(1):100–107, 2007.

[21] Haeyong Kang, Rusty John Lloyd Mina, Sultan Rizky Hikmawan Madjid, Jaehong Yoon, Mark Hasegawa-Johnson, Sung Ju Hwang, and Chang D Yoo. Forget-free continual learning with winning subnetworks. In *Int. Conf. Mach. Learn.*, pages 10734–10750. PMLR, 2022.

[22] Minsoo Kang, Jaeyoo Park, and Bohyung Han. Class-incremental learning by knowledge distillation with adaptive feature consolidation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 16071–16080, 2022.

[23] 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. *Proc. the national Academy of Sciences*, 2017.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] Zhizhong Li and Derek Hoiem. Learning without Forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[26] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics Training: Multi-Class Incremental Learning without Forgetting. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[27] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. In *Adv. Neural Inform. Process. Syst.*, 2017.

[28] Martin Mundt, Iuliia Pliushch, Sagnik Majumder, Yongwon Hong, and Visvanathan Ramesh. Unified probabilistic deep continual learning through generative replay and open set recognition. *Journal of Imaging*, 8(4):93, 2022.

[29] Xing Nie, Shixiong Xu, Xiyan Liu, Gaofeng Meng, Chunlei Huo, and Shiming Xiang. Bilateral memory consolidation for continual learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 16026–16035, 2023.

[30] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental Classifier and Representation Learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017.

[31] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, , and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *Int. Conf. Learn. Represent.*, 2019.

[32] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[33] Kaushik Roy, Christian Simon, Peyman Moghadam, and Mehrtash Harandi. Subspace distillation for continual learning. *Neural Networks*, 2023.

[34] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[35] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *Int. Conf. Mach. Learn.*, pages 4528–4537. PMLR, 2018.

[36] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual Learning with Deep Generative Replay. In *Adv. Neural Inform. Process. Syst.*, 2017.

[37] Christian Simon, Piotr Koniusz, and Mehrtash Harandi. On Learning the Geodesic Path for Incremental Learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[38] Gido M Van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1):4069, 2020.

[39] Guang Yang, Feng Pan, and Wen-Biao Gan. Stably maintained dendritic spines are associated with lifelong memories. *Nature*, 462(7275):920–924, 2009.

[40] Fei Ye and Adrian G Bors. Learning latent representations across multiple data domains using lifelong vaegan. In *Eur. Conf. Comput. Vis.*, pages 777–795. Springer, 2020.

[41] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. In *Int. Conf. Mach. Learn.*, 2017.

[42] Da-Wei Zhou, Qi-Wei Wang, Zhi-Hong Qi, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Deep class-incremental learning: A survey. *arXiv preprint arXiv:2302.03648*, 2023.