Master of Science in Informatics at Grenoble
Master Informatique
Specialization Data Science

Knowledge Transfer for Class-Incremental Learning Without Memory

Habib Slim

August 30 2020

Research project conducted at CEA-LIST

Under the supervision of:
Dr. Adrian Popescu

Defended before a jury composed of:
Pr. Massih-Reza Amini
Pr. Franck Iutzeler
Pr. Ioannis Kanellos

August 2020

Abstract

Incremental learning enables artificial agents to learn from sequential data. While important progress was made by introducing this paradigm to deep neural networks, incremental learning remains a very challenging problem. This is particularly the case when no memory of past data is allowed, in which case catastrophic forgetting has a stronger effect. We tackle class-incremental learning without memory by adapting prediction bias correction, a method which makes predictions of past and new classes more comparable. It was proposed when a memory is allowed and cannot be directly used without memory, since samples of past classes are required. We introduce a two-step learning process which allows the transfer of bias correction parameters between reference and target datasets. Bias correction parameters are first optimized offline on reference datasets with access to an associated validation memory. The obtained correction parameters are then transferred to target datasets, for which no memory is available. The second contribution is to introduce a finer modeling of bias correction by learning its parameters per incremental state instead of the usual past vs. new class modeling. Our proposed dataset knowledge transfer scheme is applicable to any incremental method operating without memory. We test its effectiveness by applying it to four existing methods. Evaluation with four target datasets and different configurations shows consistent improvement, with practically no computational and memory overhead.

Acknowledgements

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program "Investissement avenir". This master thesis could not have been completed without the unwavering support of my supervisor Dr. Adrian Popescu, and the help and assistance of Eden Belouadah. I sincerely thank them for their guidance, their patience and their willingness to share their knowledge throughout my internship.

Contents

List of Figures

List of Tables

— 1 —
Introduction

Incremental learning enables artificial agents to learn in dynamic environments in which data is presented in streams, while preserving previously acquired knowledge. This type of learning is needed when access to past data is limited or impossible, but is affected by catastrophic forgetting [37] - a phenomenon consisting in a drastic performance drop for previously learned information when ingesting new data.

In this work, we focus on class-incremental learning for computer vision, a specific setting of incremental learning in which image classes are split into multiple states and sequentially fed to a learning agent. Works such as [45, 11, 21, 55, 59, 14] alleviate the effect of forgetting by replaying past data samples when updating class-incremental models. When such a memory is allowed, incremental learning becomes an instance of imbalanced learning [18], in which new classes are naturally favored as they are represented by a larger number of images. As a result, various correction methods tackling classification bias have been successfully introduced in [11, 55, 6, 59].

While important progress was made when a fixed memory is allowed, this is less the case for class-incremental learning without memory. This last setting is more challenging and generic since no storage of past samples is allowed. In absence of a rehearsal memory, existing methods become variants of *Learning without Forgetting* (LwF) [29] with different formulations of the distillation term, or added regularization constraints. Importantly, bias correction methods become inapplicable without access to past classes samples.

Our main contribution is to enable the use of bias correction methods, such as the *BiC* layer from [55], in class-incremental learning without memory. We thus mainly focus on bias correction, as it is both simple and effective in incremental learning with memory [9, 36]. Authors of *BiC* [55] use a validation memory which stores samples of past classes to optimize parameters. Here, we learn correction parameters "offline" on a set of reference datasets and then transfer them to target datasets. While reference and target datasets follow different data distributions, we hypothesize that optimal bias correction parameters are stable enough to be transferable between them. Our proposed method is applicable to any class-incremental learning method, as it only requires the availability of raw predictions provided by deep models.

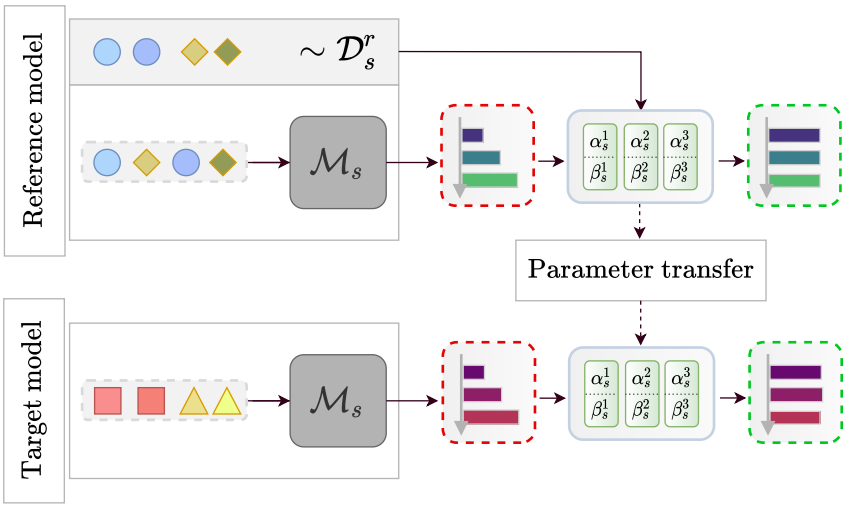


Figure 1.1 – Illustrating our proposed transfer-based bias correction method. We propose to learn a bias correction layer on reference models and to transfer it to target models during evaluation.

The second contribution of this work is to refine the definition of the bias correction layer introduced in [55]. The original formulation considers all past classes equally in the correction process. With [36], we hypothesize that the degree of forgetting associated to past classes depends on the initial state in which they were learned. Consequently, we propose "adaptive BiC" ($adBiC$), an optimization procedure which learns a pair of parameters per incremental state, instead of a single pair of parameters as proposed in [55].

We provide a comprehensive evaluation of our method by applying it on top of four backbone class-incremental learning methods. Four target visual datasets with variable domain shift with respect to reference datasets, and varying numbers of incremental states are considered. A significant improvement in top-1 accuracy is obtained for almost all tested configurations. Importantly, the additional memory needs are negligible since only a compact set of correction parameters is stored.

— 2 —
Background

We first provide a formalization of the class-incremental learning setting we consider in Section 2.1, and we introduce other related scenarios. We then describe the challenges associated with learning multiple tasks sequentially in Section 2.2.

2.1 Problem setting

2.1.1 Description

A large number of problem settings have been proposed for class-incremental learning in the literature [51]. Some works consider disjoint sets of classes for each task to be learned [29, 11, 21, 14], while others allow subsequent tasks to share classes [4, 5], effectively blurring or even removing task boundaries. Some works introduced models to be trained on online data streams [4, 3, 24], while most of other works allow samples to be seen more than once. Overall, the realism and relevancy of these proposed settings greatly vary.

In this work, we consider the offline-disjoint scenario which is the most common in the class-incremental learning literature. Our approach mainly differs to a lot of other bias correction methods in the fact that we do not allow for the use of a rehearsal memory as an additional constraint.

Formalization. A class-incremental learning problem is composed of a sequence of S states (or tasks). The first state is referred to as the initial state, and the $S-1$ remaining states are incremental states. In the s^{th} state, a set of P_s new classes is learned.

We suppose that we have access to sets of images $\mathcal{X}$ and labels $\mathcal{Y}$. A model $\mathcal{M}_1$ is initially trained on a dataset $\mathcal{D}_1 = \{(\boldsymbol{x}, y) \mid \boldsymbol{x} \in X_1^j, y \in Y_1^j;\ j \in P_1\}$, where $X_1 \subseteq \mathcal{X}$ and $Y_1 \subseteq \mathcal{Y}$ are the sets of training images and their associated labels. We note N_s the set of all classes seen until the s^{th} state included, such that: $N_s = N_{s-1} \cup P_s = P_1 \cup P_2 \cup ... \cup P_{s-1} \cup P_s$. Furthermore, all tasks are disjoint and we have: $P_i \cap P_j = \varnothing \ \ \forall i, j \in [\![1, S]\!], i \neq j$.

$\mathcal{M}_s$ is updated with an incremental learning algorithm $\mathcal{A}$ using training images from $\mathcal{D}_s$. Note that in class-incremental learning, the model does not have knowledge of the task on which it is evaluated. $\mathcal{D}_s$ includes only new classes samples, but $\mathcal{M}_s$ is evaluated on all classes seen so far ($j \in N_s$). In Figure 2.1, this class-incremental learning process is illustrated.

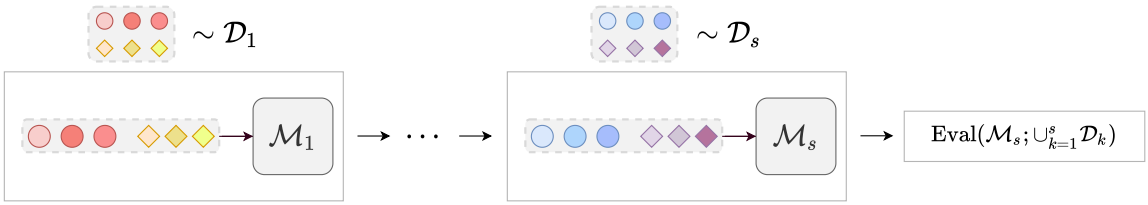


Figure 2.1 – Class-incremental learning process illustrated after s states. For each state, the model is trained on a subset of all classes of a dataset, and evaluated on all classes seen so far after training.

This makes the evaluation prone to catastrophic forgetting due to the lack of past exemplars [9, 36].

Assumptions. Additionally, we consider here a class-incremental setting operating under the following assumptions:

- **no memory.** Most methods allow for the storage of a small memory preserving knowledge from past tasks, often in the form of a limited amount of samples from seen classes. In this work, we consider a setting in which no memory of past samples is allowed, which is considerably more challenging [9].
- **pre-training.** Some works allow for the pre-training of incremental models on a large amount of initial data (generally half of the size of the training set), effectively adding a larger task at the beginning of the learning sequence [14, 57, 22]. Here, we consider that all models are trained from scratch, and that we have $|P_1| = \ldots = |P_s|$.
- **parameter growth.** Some methods allow for a significant growth of the number of parameters added to the model after each state. Here, following [45], we consider that an incremental learner should have bounded computational requirements (or very slowly growing).
- **offline training.** Finally, following a largely adopted setting in previous works, we allow for unlimited access to samples from the active state. Multiple parameter updates on a single sample are therefore allowed during training.

Assumptions made when designing a continual learning algorithm greatly vary, which makes direct comparison between methods particularly difficult. In this work, we propose to use a simple setting while restricting the use of memory or extra parameters.

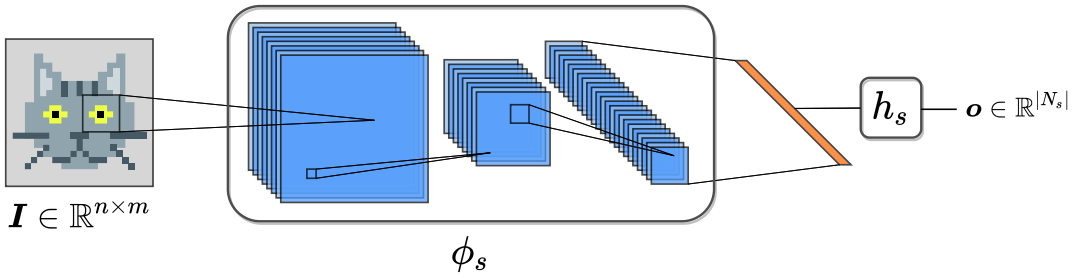


Figure 2.2 – Illustrating the notations used with a typical CNN incremental model in the s^{th} state. The feature extractor ϕ_s maps an image $\boldsymbol{I}$ to a feature vector (in orange). The classifier function h_s maps this representation to a prediction vector $\boldsymbol{o}$ with $|N_s|$ components.

Classifier. We give here some notations used to describe the learned models, which we also illustrate in Figure 2.2. With each incremental model $\mathcal{M}_s$, we associate a function f_s defined as:

$$\begin{aligned} f_s \colon \mathcal{X} &\to \mathbb{R}^{|N_s|} \\ \boldsymbol{x} &\longmapsto h_s \circ \phi_s(\boldsymbol{x}) \end{aligned} \tag{2.1}$$

where $\phi_s \colon \mathcal{X} \to \mathbb{R}^d$ is the feature extractor encoding a sample $\boldsymbol{x} \in \mathcal{X}$ into a feature vector of dimension d, and $h_s \colon \mathbb{R}^d \to \mathbb{R}^{|N_s|}$ is the classifier function. The number of outputs of our classifier is thus growing with the number of seen classes, which translates into the addition of output heads to the classifier layer after each task in the context of a typical deep architecture.

In practice, classifier weights corresponding to past tasks are usually frozen when training on new tasks. Formally $\forall t \leq s,\ \forall \boldsymbol{x} \in \mathcal{X}_t$, we have:

$$[h_s \circ \phi_s(\boldsymbol{x})]\,|_{|N_t|} = h_t \circ \phi_s(\boldsymbol{x}) \tag{2.2}$$

Where $v|_k$ denotes the vector v truncated to its first k elements. Indeed, when no access to past class features is possible, updating decision boundaries corresponding to past classes is likely to result in a loss of performance on past tasks.

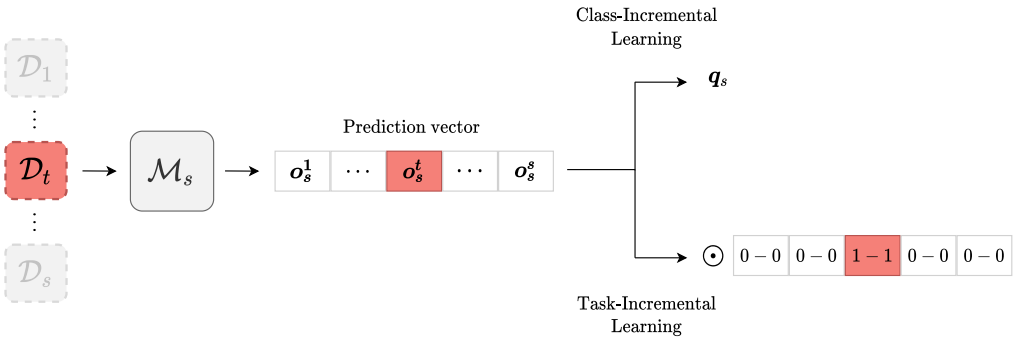


Figure 2.3 – Comparing the task-incremental learning paradigm to class-incremental learning. In task-incremental learning, when evaluating on a task $\mathcal{D}_t$ after s tasks, the resulting softmax prediction vector $\boldsymbol{q_s}$ is masked so that only output activations corresponding to classes from $\mathcal{D}_t$ are non-zero.

2.1.2 Related scenarios

We briefly present here some scenarios related to class-incremental learning, and provide some insights into their main similarities and differences.

Transfer learning. In transfer learning, given a set of source tasks $\mathcal{T}_1, \ldots, \mathcal{T}_{n-1}$ and a target domain $\mathcal{T}_n$, the aim is to train a prediction function f on $\mathcal{T}_n$ using the knowledge from previous tasks [40]. Class-incremental learning can thus be seen as a transfer learning problem with additional constraints, in which the resulting prediction function f is also evaluated on previous tasks.

Multi-task learning. In [58], the authors define multi-task learning as a problem in which given a set of tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$, the aim is to learn the n tasks simultaneously by leveraging knowledge contained in all of them. Class-incremental learning essentially replaces the simultaneous aspect of MTL with a sequential learning of tasks. Knowledge priors can still be propagated through tasks, with the use of mechanisms like distillation and rehearsal (as seen later in Section 3).

Task-incremental learning. Task-incremental learning is one of the closest scenarios to class-incremental learning explored in the literature [33, 35]. The main setup and assumptions are identical, with the only difference that task-incremental models are provided with an oracle access to a task identifier at inference. Given a classifier function f_s learned in a task-incremental setting, predictions for a test sample $\boldsymbol{x} \sim \mathcal{D}_t$ can thus simply be masked with:

$$f_s(\boldsymbol{x}) \odot \boldsymbol{m_t}, \;\; \boldsymbol{m_t} \in \{0, 1\}^{|N_s|} \tag{2.3}$$

where t is the task identifier associated to sample $\boldsymbol{x}$, $\boldsymbol{m_t}$ is the corresponding binary mask on the output scores, and $\odot$ is the Hadamard product (see Figure 2.3). In practice, this eliminates two of the most prominent problems of class-incremental learning: inter-task confusion and task-recency bias (see following Section 4.2).

Supervised learning. Finally, supervised learning can naturally be related to continual learning and constitutes an obvious upper-bound of incrementally learned models when training on the same data distribution. We illustrate this here by detailing the varying objectives of continual and supervised learning, in the context of the classification.

We consider samples from a dataset $\mathcal{D}$ and a class-incremental setting with S states. Following notations from Section 2.1.1 we have: $\mathcal{D} = \bigcup_{s=1}^{S} \mathcal{D}_s$.

Let $\mathcal{H}_S$ be the functional space of classifiers from samples $\boldsymbol{x} \in \mathcal{X}$ to the output activation space $\mathbb{R}^{|N_S|}$. Let ℓ be an error function defined as:

$$\begin{aligned} \ell \colon \mathcal{H}_S \times \mathcal{X} \times \mathcal{Y} &\to \{0, 1\} \\ f, \boldsymbol{x}, y &\longmapsto \mathbb{1}_{\{y\}} \left(\arg\max_{c \in N_S} f(\boldsymbol{x})_c \right) \end{aligned} \tag{2.4}$$

where $\mathbb{1}$ denotes the indicator function. In a supervised learning setting, we search for a function $\hat{f}$ that is such that:

$$\hat{f} = \inf_{f \in \mathcal{H}_S} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[\ell(f; \boldsymbol{x}, y)] \tag{2.5}$$

whereas in an incremental learning setting, we *explicitly* and successively search for functions $\hat{f}_s$ defined in each of the S states as:

$$\hat{f}_s = \inf_{f \in \mathcal{H}_s} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}_s}[\ell(f; \boldsymbol{x}, y) + \Omega(f, \mathcal{K}_{s-1}; \boldsymbol{x}, y)] \tag{2.6}$$

with the *implicit*[1] goal of reaching a function $\hat{f}_S$ that also minimizes the objective of (2.5). In order to reach this implicit goal, an additional term utilizing knowledge $\mathcal{K}_{s-1}$ from the previous state is added to the main objective. In a memoryless scenario, this knowledge is reduced to the classifier learned in the preceding state. The function Ω can then, as an example, constrain the drift of activations across states:

$$\Omega(f, f_{s-1}; \boldsymbol{x}, y) = \|f_{s-1}(\boldsymbol{x}) - f(\boldsymbol{x})\|_F \tag{2.7}$$

This is analogous to knowledge distillation [20], and is a common way to reduce forgetting in continually learned models [29, 13, 21].

[1] In works using meta-learning to tackle class-incremental learning, the global objective of (2.5) can be formulated as a meta-loss and is then explicitly optimized for.

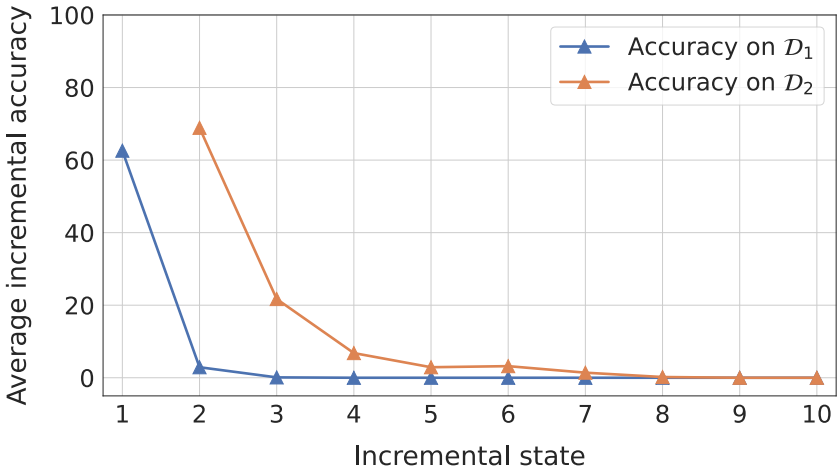


Figure 2.4 – Average top-1 accuracies on the first and second split of CIFAR-100 (split in ten folds), of a fine-tuning model trained incrementally. On both tasks and after only a few states, the model performance suffers from catastrophic forgetting.

2.2 Known challenges

Various challenges associated with incremental learning have been identified, often through empirical studies of fine-tuned models [49, 36]. We discuss here the main challenges of learning classification models incrementally (or continually).

Catastrophic forgetting. Catastrophic forgetting or *catastrophic inference* is an intensely studied phenomenon which received a lot of attention in the early literature on deep neural networks [37, 44, 16]. It is widely described as a drastic loss in performance on previous tasks when a neural network learns a new task. This performance loss is analogous to the loss of previously learned information in biological systems, and is therefore generally referred to as "forgetting". However, while forgetting is mostly gradual in natural cognitive systems, neural networks which do not make use of any stability constraints will exhibit abrupt forgetting patterns on previously acquired knowledge when new information is presented to them [16].

To illustrate this point, we incrementally train a ResNet-18 [19] network on the CIFAR-100 dataset, which we split into ten states. Following our proposed setup in Section 2.1.1, each state contains 10 classes and states do not overlap. We use a common fine-tuning strategy in which the feature representation ϕ is updated in all states, but the classifier h is frozen for outputs corresponding to past classes in order to preserve previously learned decision boundaries [8, 36].

In Figure 2.4, we plot the single-task accuracy on the first task (blue) and the second task (orange), evaluated on $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. While the model performs well initially on each of the two tasks, the performance immediately drops below 5% after the second state for the first task. For the second task, this forgetting is also abrupt - although more gradual. This is a common pitfall of incrementally learned models, and is not exclusive to discriminative models.

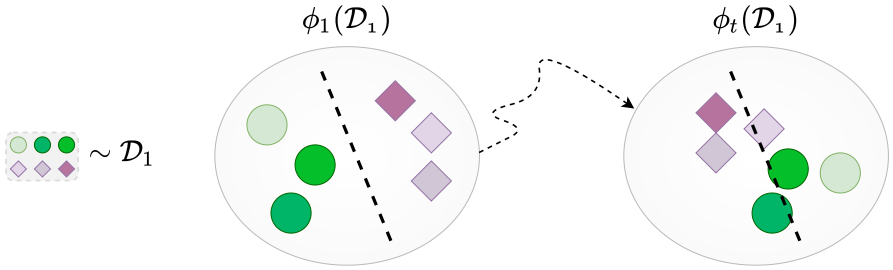


Figure 2.5 – Illustrating the representation drift phenomenon in incrementally learned models, when retraining on past samples is prohibited. After $t-1$ states, the learned feature representation ϕ_t is no longer compatible with the decision boundaries learned in the initial state for samples from the first split $\mathcal{D}_1$.

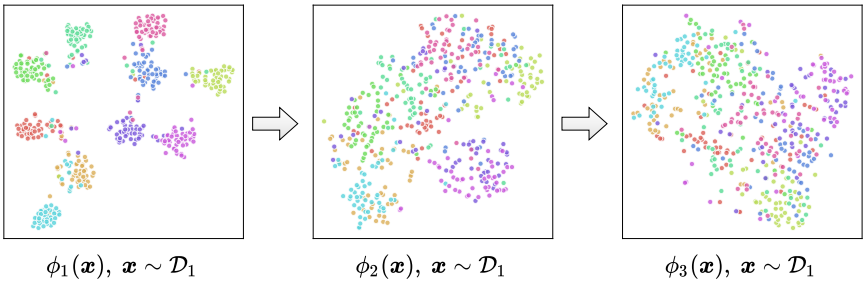
In order to prevent catastrophic forgetting, incremental models must use methods which enforce the stability of previously acquired knowledge, while allowing the acquisition of new information. This problem is known as the stability-plasticity dilemma [38].

Various culprits have been identified as factors of catastrophic forgetting in the literature [21, 36]. Here, we focus on three of the main problems of incremental supervised learning.

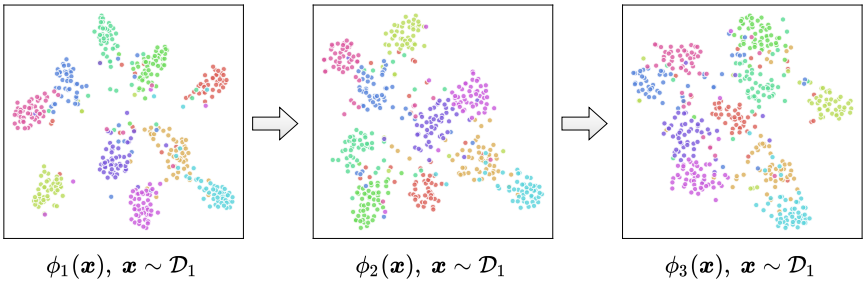
Representation drift. When updating a model on a new task $\mathcal{T}_n$ after learning a set of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_{n-1}$, the representation of past tasks is degraded as the model does not have full access to past tasks samples.
Formally, for a class-incremental learning process with at least $s+1$ states and $\boldsymbol{x} \sim \mathcal{D}_s$, there is no guarantee to have: $\|\phi_s(\boldsymbol{x}) - \phi_{s+1}(\boldsymbol{x})\|_F \leq \epsilon$. Therefore, learned (frozen) decision boundaries may no longer be compatible with representation ϕ_{s+1}. The updated feature extractor ϕ_{s+1} may not even properly enable the separation of classes from P_s, in which case incremental learning strategies focusing on updating classifier weights become ineffective.

In Figure 2.5, representation drift is illustrated for samples belonging to a first subset $\mathcal{D}_1$, in an incremental learning process with t disjoint tasks. After $t-1$ states, the updated feature extractor ϕ_t is no longer compatible with the decision boundary initially learned, which causes misclassifications for classes from the first task.
Note that in this example, recovering a perfect accuracy on the first task would be possible without having to update the feature representation ϕ_t by correcting the learned decision boundaries for the first two classes (which is equivalent to updating h_t). In practice however, the final feature projection of samples from $\mathcal{D}_1$ may not even properly separate classes from P_1 anymore.

To confirm this, we train a model across three splits of ImageNet-100 and extract the features of samples belonging to the first split. We visualize these features in Figure 2.6 for a simple fine-tuning method (top row) and for Lucir (bottom row). With fine-tuning, after learning a second task, the updated feature representation ϕ_2 is no longer able to separate data from the first task, and most structure is immediately lost.



(a) Fine-tuning



(b) LUCIR [21]

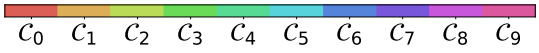


Figure 2.6 – t-SNE [52] visualization of the embedding space of a ResNet-18 [19] model trained with fine-tuning (top) and LUCIR (bottom) on IMAGENET-100 across three tasks, for samples from the first ten classes learned.

We compare this with the same visualization for LUCIR (bottom row), which makes use of a stability constraint in the feature representation of samples across models from different states. Structure is preserved across states, although inter-class separations are defined more clearly with ϕ_1 compared to subsequent models.

Some solutions to representation drift explored in previous works include:

- penalizing changes in representations [21, 29, 45]
- updating the classifier layer with a memory of past samples [11, 23, 45]
- predicting the drift of feature vectors [57, 23]

In the memoryless scenario we explore here, this phenomenon is amplified as performing parameter updates on past samples becomes impossible.

Inter-task confusion. Incremental models are evaluated on classes from all tasks. However, during training, models are updated jointly only with classes belonging to a single task. Therefore, the model is unable to optimally learn cross-task features [49], that is features enabling the model to discriminate between classes belonging to different tasks.

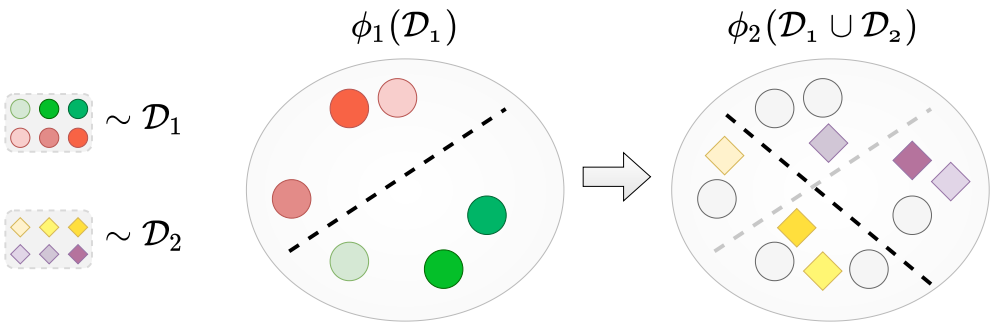


Figure 2.7 – Illustrating inter-task confusion when retraining on past samples is prohibited. In the second state, while we do learn a feature representation ϕ_2 enabling the separation of classes from $\mathcal{D}_2$, the model cannot differentiate between circles and diamonds as they were seen in different tasks.

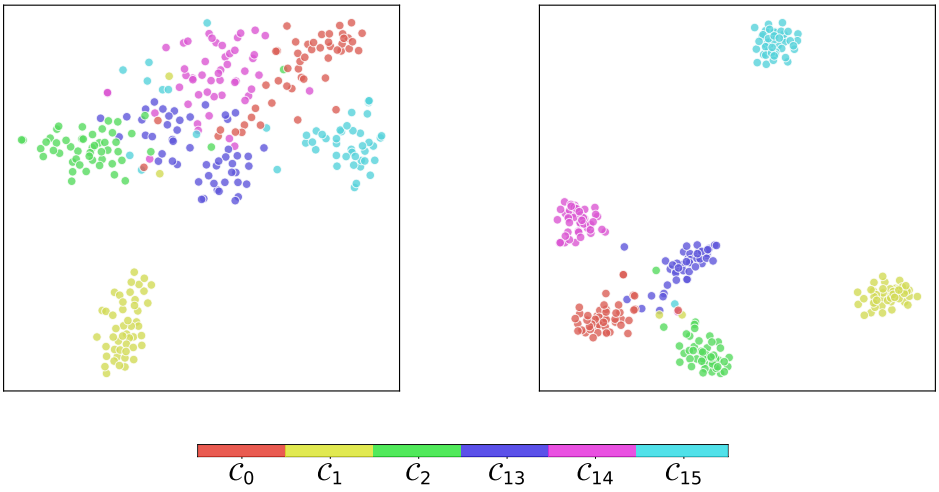


Figure 2.8 – t-SNE [52] visualization of features extracted with a model trained incrementally with LUCIR over two tasks (left) and with a model trained jointly with all data (right), on a fine-grained subset of ImageNet. Three classes from the first task ($\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2 \sim P_1$) and from the second task ($\mathcal{C}_{13}, \mathcal{C}_{14}, \mathcal{C}_{15} \sim P_2$) are selected to highlight inter-task confusion.

In Figure 2.7, the problem of inter-task confusion in incrementally learned models is illustrated. With a supervised classification loss in a memoryless scenario, we are likely to properly separate classes belonging to the same task. However our model has no incentive to learn a feature space ϕ_2 able to separate classes from different tasks. Indeed, in a disjoint class-incremental setting without memory, samples from $\mathcal{D}_1$ and $\mathcal{D}_2$ cannot be used jointly to optimize the objective function.

This important shortcoming of incrementally learned models is experimentally verified in Figure 2.8, in which the representations of samples from two different tasks $\mathcal{D}_1$ and $\mathcal{D}_2$ (left) are compared to their representations when the model is jointly trained using $\mathcal{D}_1 \cup \mathcal{D}_2$ (right). While the incrementally trained model manages to discriminate classes within the same task (although some ambiguities are noticeable for $\mathcal{C}_{13}$ and $\mathcal{C}_{14}$), classes from different tasks clearly overlap. With the jointly learned representation, classes are clearly separated with very little inter-class ambiguity. Note that we trained these models on a fine-grained subset of ImageNet-100 and selected highly ambiguous classes to better highlight these effects.

In more general class-incremental learning settings, solutions proposed to inter-task confusion include:

- adding margin losses in combination with reserved samples to alleviate confusion between classes [21, 14]
- hallucinating features from future classes and maximizing distance to hallucinated (incorrect) future class proxies [15]. This regularization method is particularly clever, but requires access to another modality for future classes (in [15], the text descriptions of future classes are used to hallucinate probable features), in combination with a generator network - which partially breaks the common class-incremental learning assumptions.

Again, this problem is even more prominent without access to a memory as new and past samples are never used together in the same batch. In [49], authors show that cross-task features are responsible for a significant chunk of the performance gap between class-incremental methods and standard joint training.

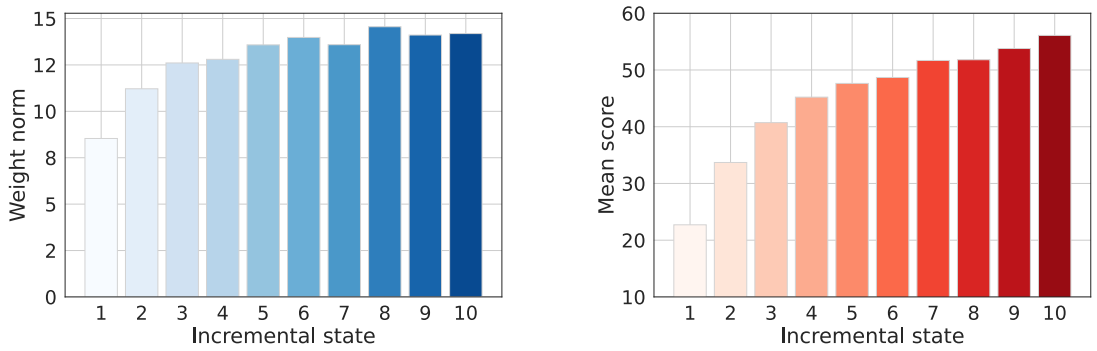


Figure 2.9 – Frobenius norm of classifier weights dedicated to classes from each state (left) and mean prediction scores per state (right) for a LUCIR model trained on CIFAR-100 with 10 splits.

Task-recency bias. Finally, task-recency bias is a phenomenon recognized as one of the main sources of catastrophic forgetting in neural networks [55, 6, 36], and is the one we attempt to tackle in this work.

The problem of task-recency bias can be summarized as follows. When learning multiple tasks sequentially, there is no guarantee that the magnitude of classifier scores will remain balanced across classes learned in different states. In practice, classifier scores are heavily biased towards the last task which translates in misclassifications towards the most recent classes.

In Figure 2.9, we illustrate the task-recency bias problem by plotting the Frobenius norm of class embeddings for each task (left) and the mean prediction scores per state (right) for an incremental model trained with LUCIR across $S = 10$ CIFAR-100 splits. The bias towards more recent classes is clearly evidenced by the increasing magnitude of classifier norms and prediction scores as a function of the state.

Solutions to task-recency bias proposed in the literature include:

- using a memory to correct bias by applying linear transformations to outputs [6, 55]
- retraining the classifier layer on a balanced dataset using saved exemplars [11]

However, there is a significant gap in existing works regarding bias correction when access to a memory of past samples is forbidden. In this work, we attempt to bridge this gap by proposing a method for memoryless bias correction.

— 3 —
Related Works

Incremental learning is a longstanding machine learning problem [17, 34, 50], which witnessed a strong growth in interest after the introduction of deep neural networks. Detailed reviews of existing approaches are proposed, among others, in [9, 28, 36, 41]. Here, we analyze works most related to our proposal, which tackles class-incremental learning while keeping memory and computational requirements constant, or nearly so, during the learning process.

Following assumptions made in Section 2.1.1, we focus here on class-incremental learning methods working under the disjoint assumption, in the offline formulation - that is, allowing unrestricted access to the whole dataset in each task. We focus particularly on methods which address *task-recency bias*, and were designed for class-incremental learning with memory.

In the first section, we will describe the main categories of methods used to solve the class-incremental earning problem, with some examples.

In the second section, we focus more specifically on methods relevant to this work, and on the backbone class-incremental learning methods we build upon.

3.1 Main approaches

Multiple categorizations of deep continual learning methods have been proposed in previous studies [36, 39]. Most methods are in practice hybrid and generally employ a combination of rehearsal and regularization or bias correction approaches, as highlighted in Figure 3.1. Following [36], we focus here on three of the main approaches for class-incremental learning.

Bias correction approaches. The goal of bias correction approaches is to correct the task-recency bias of incrementally learned models. Correcting this bias can involve:

- transformations of the output scores of a CNN model [6, 55, 7]
- adding a balanced fine-tuning stage to alleviate its effects [11, 49]
- or performing inference without using classifier weights [45, 21, 57]. In this case, incoming samples are classified using learned class embedding means (also referred

to as "prototypes" in the literature.). The nearest class embedding is then generally used to infer the target class of the sample.

Incremental learning methods falling into this last category usually incrementally train using classifier weights, before discarding the layer at inference and performing classification using prototypes. However, a recent approach proposes instead to completely remove the classifier layer and learn an embedding network with metric learning losses instead of classification losses [57].

Our proposal is closely related to bias correction methods, as we also employ a linear bias correction scheme derived from [55]. In order to estimate the amplitude of the task-recency bias in order to properly correct it, most methods make use of a memory containing either dataset statistics [6] or past exemplars [55, 59].

In this work, we tackle two important limitations of existing bias correction methods. First, they are inapplicable without memory because they require the presence of past class samples: we propose to transfer bias correction layer parameters between datasets to address this problem. Second, the degree of forgetting associated to past classes is considered equivalent, regardless of the initial state in which they were first learned: we refine the linear layer from [55] to improve the handling of task-recency bias.

Regularization approaches. The wide majority of class-incremental learning methods make use of an information preserving penalty [13]. This penalty is generally implemented as a loss function which reduces the divergence between the current model and models learned in preceding incremental states, in order to maximize the overall performance on all seen classes.

Regularization methods employ additional regularization terms to the objective loss function implementing this penalty, specifically designed to minimize the effects of catastrophic forgetting. These regularization terms usually aim at:

- explicitly reducing the drift of parameters important for performance on classes seen in previous states [26, 2]
- or reducing the drift of higher-level representations for past classes. This can be done:
  - at the feature level [21], in which case methods will generally make use of past exemplars to evaluate and reduce the change in feature representations of previously learned classes.
  - at the output level [29, 45], which works similarly to feature-level penalties, and is analogous to knowledge distillation [20].
  - or simultaneously at multiple representation levels [21, 14].

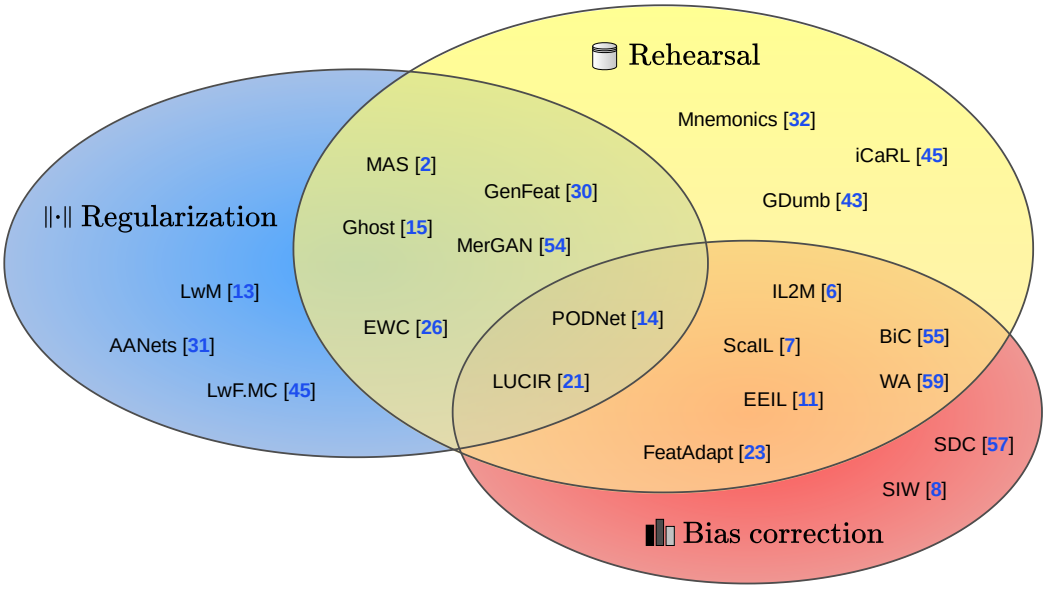


Figure 3.1 – Venn diagram of class-incremental learning methods operating over the disjoint tasks assumption. Almost all methods are mainly designed to take advantage of a rehearsal memory, or rely on pseudo-rehearsal based on synthetic samples or features.

In [39], the authors propose to discriminate between regularization terms constraining the outputs of a model - referred to as *functional* regularization methods, from regularization methods operating at intermediate levels, coined *structural* methods.

Regularization methods generally have the drawback of inducing additional memory costs, as most of them require to save an instance of the model in the previous state or a memory of feature vectors or selected exemplars, in order to evaluate representation drift. These methods are thus mostly used in combination with a rehearsal memory.

Rehearsal approaches. Rehearsal methods or *replay* methods make use of a rehearsal memory preserving knowledge from past tasks to preserve model performance when training on future tasks. These methods allow for the storage of a small memory generally containing either past tasks samples [45, 11, 21, 14, 32, 5] or features [23, 56]. In practice, this implies poor scalability when memory size is not limited.

In [53], the authors also show that although rehearsal is effective at keeping models in the first found low-loss region in early tasks, its usage can ultimately harm generalization. In this work, we consider a setting in which no memory of past samples is allowed, which is considerably more challenging [9].

Rehearsal is not directly relevant in the context of this work, but overwhelmingly used in class-incremental learning works since [45] - while very little work has been done in the memoryless scenario we consider here.

3.2 Methods

Here we focus on methods relevant to the class-incremental learning framework and closer to our assumptions, which we briefly describe. Methods directly used in this work are highlighted in blue.

LwF. Learning without Forgetting (LwF) [29] is one of the earliest works tackling catastrophic forgetting in deep neural networks. This method exploits knowledge distillation [20] to preserve information related to past classes during incremental model updates, and thus belongs to the functional category of regularization approaches introduced in Section 3.1.
In the s^{th} state, for a sample $(\boldsymbol{x}, y) \in \mathcal{D}_s$, LwF trains incremental models by adding the following distillation term to a cross-entropy loss:

$$\mathcal{L}(\boldsymbol{x}) = \sum_{k=1}^{|N^{s-1}|} \pi_k^{s-1}(\boldsymbol{x}) \log(\pi_k^s(\boldsymbol{x})) \tag{3.1}$$

where $\pi_k^{s-1}(\boldsymbol{x})$ and $\pi_k^s(\boldsymbol{x})$ are the temperature-scaled output scores from the learned classifier functions f_{s-1} and f_s respectively. As mentioned in the previous section, this implies access to a saved instance of the model in the previous incremental state. In [13], the authors build on this work by adding an attention loss to the distillation loss, extended from [48].

iCaRL. Incremental classifier and representation learning (iCaRL) [45] is the first work proposing the use of a rehearsal memory in combination with a distillation loss in the context of deep incremental learning, by building upon LwF. The authors also propose to discard the classifier layer and use a nearest-mean-of-examplars classifier.

EEIL. In [11], the authors propose to correct bias by adding a fine-tuning stage at the end of each state. This additional training stage is performed on a class-balanced dataset composed of reserved samples from current and past tasks, to alleviate the effects of task-recency bias.

Bias Correction. In [55], the authors propose a bias correction method (*BiC*), which introduces an additional layer to rebalance the raw scores of a deep incremental model. A validation set is used to optimize the parameters of this linear layer, which modifies the predictions of the deep model learned in a given incremental state. We build on this work to better handle *task-recency bias* by adding task-specific parameters in this linear layer, and by enabling its use in a memoryless scenario.

LUCIR. In [21], the authors propose to learn a unified classifier incrementally via rebalancing (Lucir). To that end, three main contributions are proposed:

- A cosine normalization layer is introduced before the softmax activation, to improve the comparability of output scores (helping *task-recency bias*). The distillation loss from LwF is slightly adapted to accommodate for this modification.

- A less-forget constraint is proposed to prevent *representation drift*. For a given training sample $\boldsymbol{x} \sim \mathcal{D}_s$, this constraint is simply formulated as:

$$\mathcal{L}(\boldsymbol{x}) = \|\phi_{s-1}(\boldsymbol{x}) - \phi_s(\boldsymbol{x})\|_F \tag{3.2}$$

- Finally, to tackle the issue of *inter-task confusion*, the authors introduce a margin ranking loss to better separate reserved samples (saved in an exemplar memory) from samples in the current task. This additional loss is thus unapplicable in a memoryless scenario. Recent benchmarks [9] show that LUCIR remains competitive even when the margin ranking loss is removed to accomodate for the memoryless scenario.

In [14], the authors build on this work by extending the less-forget constraint to be applied at shallower stages of a deep CNN, by pooling the convolutional outputs of the current and past models.

SIW. In [8], the authors propose to replay the classifier weights first learned in past tasks and to normalize them to tackle *task-recency bias*. This method is a simple baseline but is nevertheless shown to be competitive against other state-of-the art methods on large-scale datasets such as ImageNet [9].

Feature Adaptation. In [23], the authors propose to store features belonging to past tasks and to train a feature adaptation network able to make past features (extracted with ϕ_{s-1}) more compatible with the new representation ϕ_s. The adapted features from past classes are then used conjointly with features from new classes to retrain the classifier layer. Again, this work is based on a modified version of LWF, in which a cosine normalization is added before the softmax output layer.

Semantic Drift Compensation. In [57], the authors propose a class-incremental learning method operating without memory. They do not train with a classifier layer and focus exclusively on learning good feature representations in each state. To that end, new formulations of LWF and LUCIR are given, and they propose to approximate and correct the *representation drift* of past classes by measuring the drift of current task data.

Overall, as highlighted in Figure 3.1, memoryless class-incremental learning methods rely on regularization and bias correction to compensate for the absence of rehearsal. It is also worth noting that experimental results indicate that a large margin of improvement is still possible [9] for this setting, as most methods greatly suffer from catastrophic forgetting without access to past exemplars.

— 4 —
Method

Following the framework we proposed in Section 2, we introduce here our method which operates without memory on top of any class-incremental learning method providing output scores. In Section 4.1, we describe our first contribution: the introduction of a correction layer which considers bias in each state independently. In Section 4.2, we propose a simple knowledge transfer scheme to apply calibration parameters in a memoryless scenario.

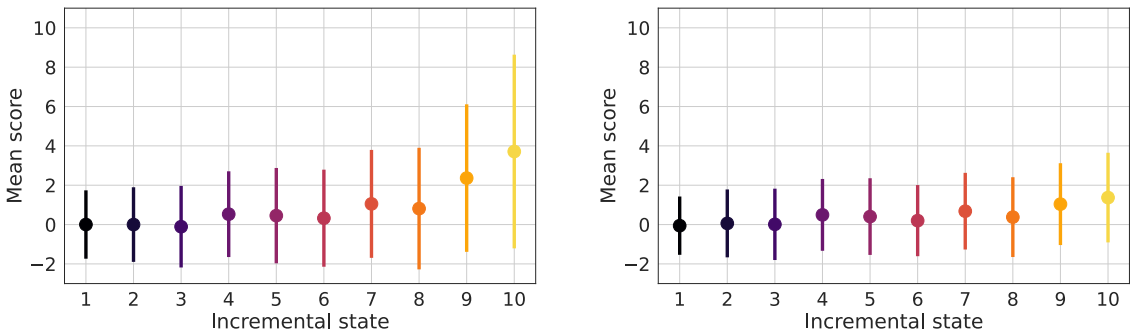
4.1 Adaptive bias correction layer

Analysis. The unavailability of past class samples when updating incremental models leads to a classification bias towards new classes [55, 59, 36]. We illustrate this in Figure 4.1 (*left*), by plotting the mean prediction scores in each state, for LUCIR and LWF models trained on CIFAR-100, the two distillation-based methods tested in this work. Models are trained without memory to fit our problem setting, with $S = 10$ splits. Figure 4.1 confirms that recently learned classes are favored, despite the use of stability constraints to counter the effects of catastrophic forgetting in both methods. New classes, learned in the last state, are particularly favored. Noticeably, the prediction profiles for LUCIR and LWF are different. With LUCIR, mean predictions per state increase from earlier to latest states, while the tendency is less clear for LWF. LWF predictions also have a stronger deviation in each state, while LUCIR mean scores are very stable across states, which makes LUCIR a better candidate for bias correction.

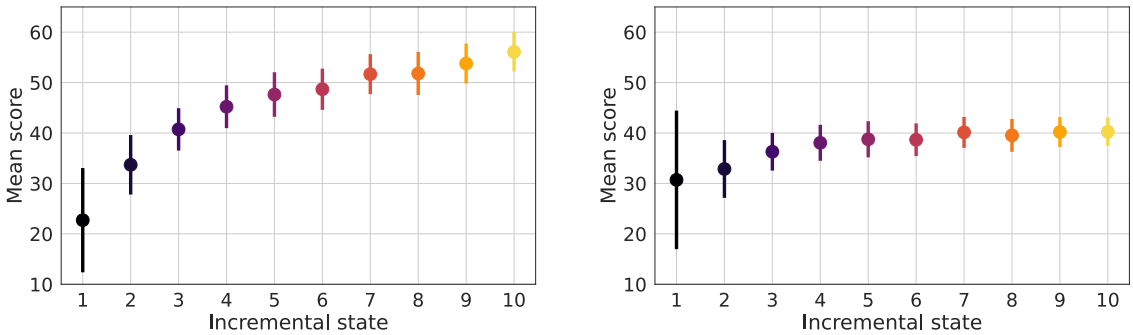
***BiC* layer.** Among the methods proposed to correct bias, the linear layer introduced in [55] is interesting for its simplicity and its performance in a large number of settings [9, 36]. This layer is defined in the s^{th} state as:

$$BiC(\boldsymbol{o_s^k}) = \begin{cases} \boldsymbol{o_s^k} & \text{if} \quad k \in [\![1,\ s-1]\!] \\ \alpha_s \boldsymbol{o_s^k} + \beta_s \cdot \mathbf{1} & \text{if} \quad k = s \end{cases} \tag{4.1}$$

where $\boldsymbol{o_s^k} \in \mathbb{R}^{|N_s|}$ are the raw scores (before softmax) of classes first seen in the k^{th} state, obtained with an incremental model $\mathcal{M}_s$; (α_s, β_s) are the bias correction parameters in the s^{th} state, and $\mathbf{1}$ is a vector of ones. Equation (4.1) rectifies the raw predictions of new classes learned in the s^{th} state to make them more comparable to those of past classes.



(a) LwF [29]



(b) LUCIR [21]

Figure 4.1 – Mean prediction scores and associated standard deviations for CIFAR-100 classes grouped by state at the end of an IL process with $S = 10$ states, for LWF and LUCIR, before (left) and after (right) calibration via our method.

The deep model is first updated using $\mathcal{D}_s$, which contains new classes for this state. The model is then frozen and calibration parameters (α_s and β_s) are optimized using a validation set made of a balanced mix of new and past classes samples.

adBiC. Equation (4.1) is not applicable in class-incremental learning without memory, the scenario explored here, as optimizing the layer requires access to past class samples. Furthermore, Figure 4.1 (*left*) shows that the mean scores of classes learned in different incremental states greatly vary, which confirms that the amount of forgetting is uneven across past states: ideally, the bias in the predictions of incrementally learned models should thus be corrected with state-specific parameters.

Following these observations, we define an adaptive version of *BiC* which rectifies the predictions $\boldsymbol{o_s^k}$ in the s^{th} state with:

$$adBiC(\boldsymbol{o_s^k}) = \alpha_s^k \boldsymbol{o_s^k} + \beta_s^k \cdot \mathbf{1} \; ; \quad \forall k \in [\![1, s]\!] \tag{4.2}$$

where α_s^k, β_s^k are applied in the s^{th} state to classes first learned in the k^{th} state. While Equation (4.1) treats all past classes predictions equally, Equation (4.2) adjusts prediction scores depending on the state in which classes were first encountered in the incremental process. In Figure 4.2, we illustrate our proposition and its differences with BiC.

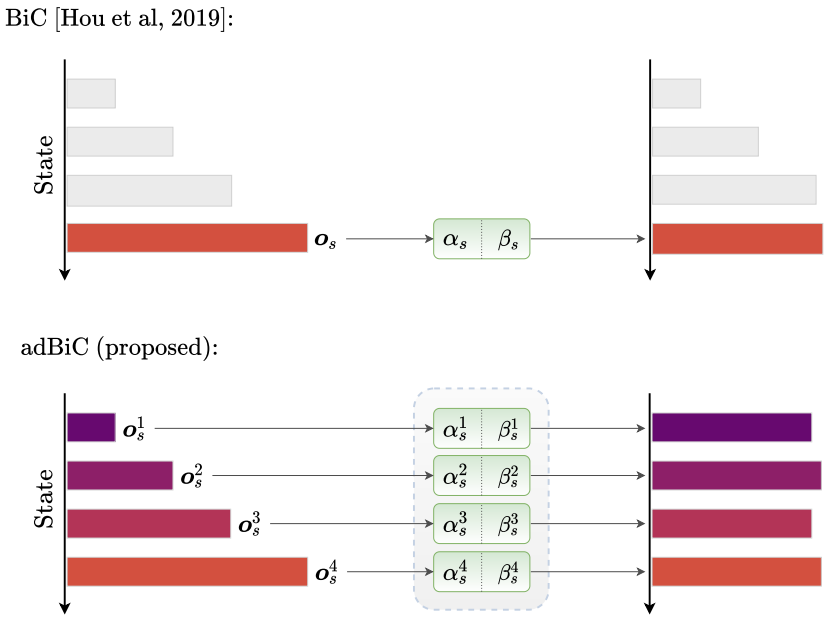


Figure 4.2 – Comparing our proposed linear bias correction layer to the one proposed in [55]. Instead of only adjusting the prediction scores of the last state, we learn a pair of coefficients for each state to better adjust the prediction profile. Furthermore, all coefficients are jointly optimized in each state.

Note that each α_s^k, β_s^k pair is shared between all classes first learned in the same state.

These parameters are optimized on a validation set using the cross-entropy loss, defined for one data point $(\mathbf{x}, y)$ as:

$$\mathcal{L}(\boldsymbol{q_s}, y) = -\sum_{k=1}^{s}\sum_{i=1}^{|P_k|} \delta_{y=\widehat{y}} \ \log\left(q_{s,i}^k\right) \tag{4.3}$$

where y is the ground-truth label, $\widehat{y}$ is the predicted label, δ is the Kronecker delta, and $\boldsymbol{q_s}$ is the softmax output for the sample corrected via Equation (4.2), defined as:

$$\begin{aligned} \boldsymbol{q_s} &= \sigma\left(\left[\ adBiC(\boldsymbol{o_s^1}) \quad ; \ \ldots \ ; \ adBiC(\boldsymbol{o_s^s})\ \right]\right) \\ &= \sigma\left(\left[\ \alpha_s^1\boldsymbol{o_s^1} + \beta_s^1 \cdot \mathbf{1} \ ; \ \ldots \ ; \ \alpha_s^s\boldsymbol{o_s^s} + \beta_s^s \cdot \mathbf{1}\ \right]\right) \end{aligned} \tag{4.4}$$

where σ is the softmax function.

All α_s^k, β_s^k pairs are then optimized using validation samples from classes in N_s. We compare *adBiC* over *BiC* for our class-incremental learning setting in the evaluation section, and show that the adaptation proposed here has a positive effect.

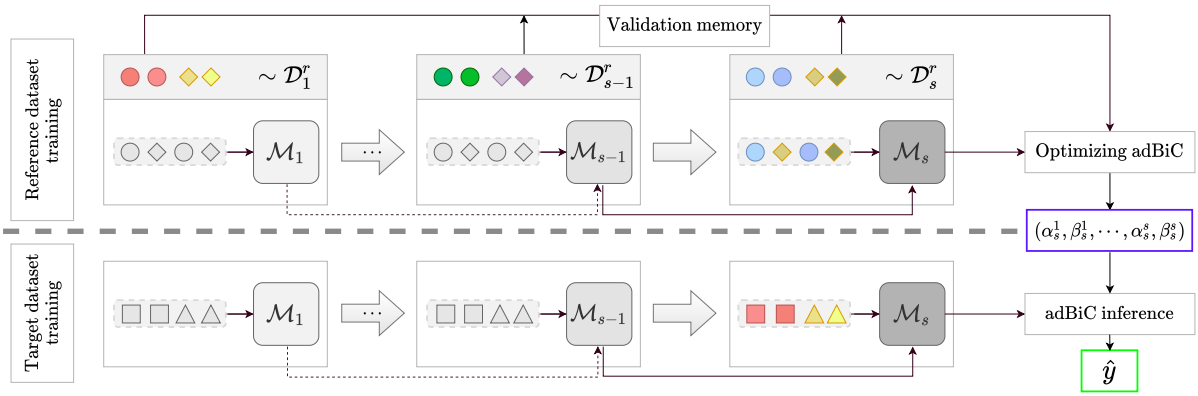


Figure 4.3 – Illustration of our proposed transfer method, depicting states from 1 to s for reference and target datasets $\mathcal{D}^r$ and $\mathcal{D}^t$. States from 1 to $s-1$ are faded to convey the fact that learned knowledge in these states is affected by forgetting, alongside reference models $\mathcal{M}^r$ and target models $\mathcal{M}^t$. The validation memory used for optimization of *adBiC* parameters is represented on top of the reference training states, in grey. After the transfer of correction parameters learned on reference datasets, the outputs of the current target model $\mathcal{M}_s^t$ are rebalanced, and a corrected prediction $\widehat{y}$ is produced.

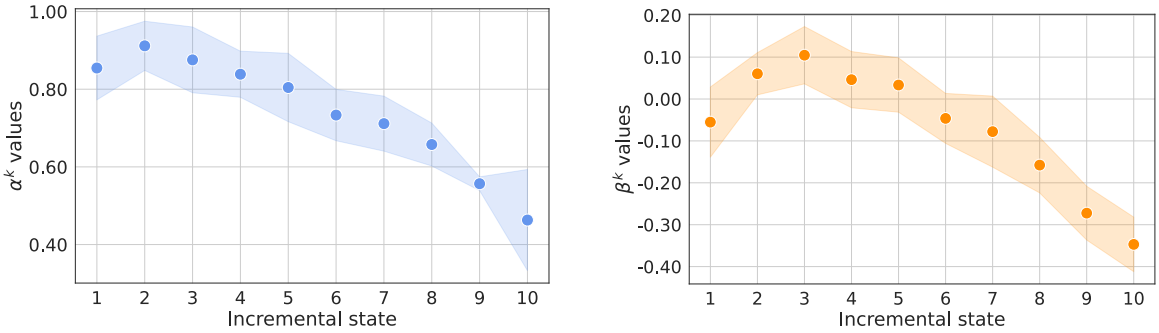
4.2 Knowledge transfer between datasets

Analysis. In the original setup proposed in [55], the optimization of α and β parameters is impossible without memory since exemplars of past classes are unavailable. To circumvent this problem, we hypothesize that optimal values of these parameters can be transferred across datasets, under the assumption that calibration profiles remain stable. To illustrate this hypothesis, we provide in Figure 4.4 *adBiC* parameters values averaged across $R = 10$ reference datasets[1]. We plot α^k and β^k values learned after $S = 10$ incremental states, using LwF [29] and Lucir [21], with standard deviations. The parameter ranges from Figure 4.4 confirm that, while optimal values do seem to vary across datasets, this variation is rather low and calibration profiles remain similar. This opens up the possibility of a parameter transfer between reference and target datasets.

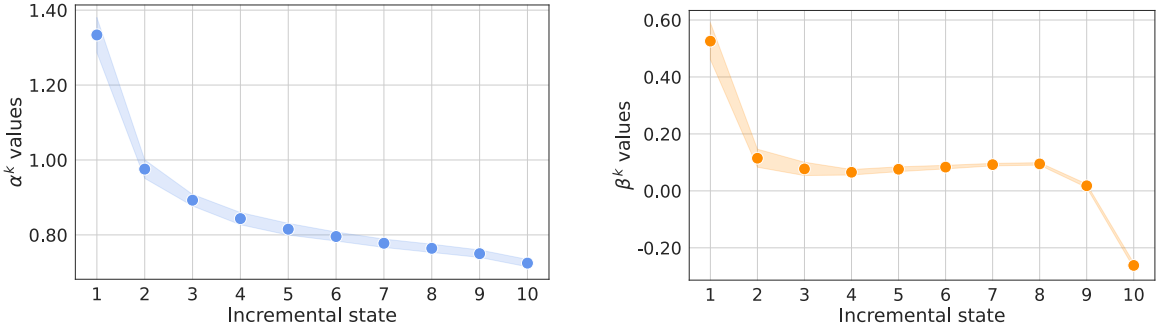
Bias correction without memory. We propose to transfer learned bias correction parameters between reference and target datasets, noted $\mathcal{D}^r$ and $\mathcal{D}^t$ respectively. In Figure 4.3, we illustrate our approach. The upper part of the figure describes the incremental training of reference models $\mathcal{M}^r$ on reference datasets $\mathcal{D}^r$. In each state, a validation memory is set aside. At the end of each state, *adBiC* parameters are optimized following Algorithm 1 of Figure 4.6, to obtain the correction parameters for this state $\boldsymbol{\theta}_s^r$.

After training a model $\mathcal{M}^r$ in each state, a set of *adBiC* parameters are optimized following Equation (4.3) using a validation set containing samples from all past classes. We remark that the use of this validation set would violate the class-incremental setting proposed in Section 2 if the evaluation was performed using the reference models $\mathcal{M}^r$, as we revisit past classes samples in order to optimize parameters from Equation (4.2).

[1]For additional details about how these reference datasets are built, please refer to Section 5.1.3



(a) LwF [29]



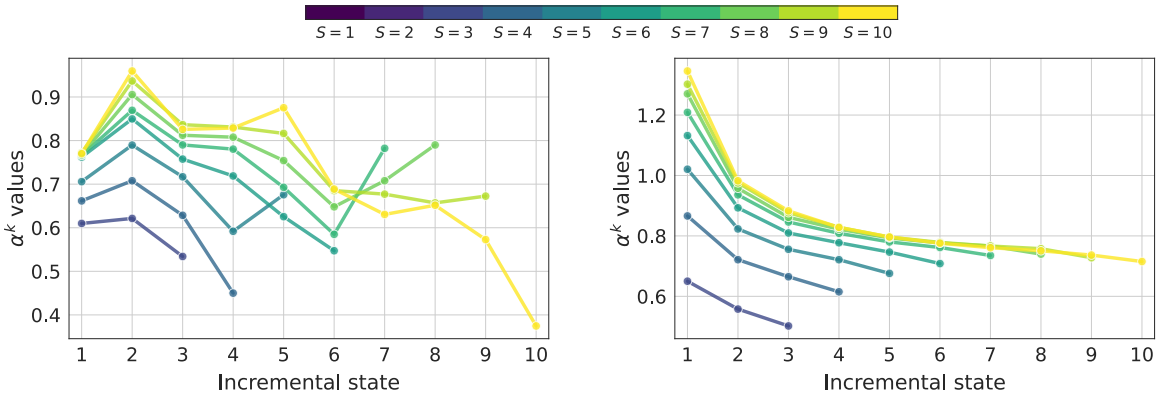
(b) LUCIR [21]

Figure 4.4 – Averaged α^k (left) and β^k (right) values computed for $R = 10$ reference datasets using LwF and LUCIR, at the end of an incremental process with $S = 10$ states. Standard errors for each coefficient are shaded.

We then store bias correction parameters optimized for reference datasets in order to perform transfer towards target datasets without using a memory. For each incremental state, we compute the average of α and β values over all reference datasets. The obtained averages are then used for score correction on target datasets, following the procedure described in Algorithm 2 of Figure 4.6.

Backbone incremental models for $\mathcal{M}^r$ are trained without memory in order to simulate the incremental setting of target models $\mathcal{M}^t$. Note that we make no assumptions on the source distributions of datasets $\mathcal{D}^r$ and $\mathcal{D}^t$, but we empirically show later in Section 5.1 that similarity of these distributions impacts the transferability of correction parameters.

Memory costs. The memory needed to store transferred parameters is negligible since we need $S(S+1)-2$ floats for each dataset and S value. For $S = \{5, 10, 20\}$ states, we thus only store 28, 108 and 418 floating-point values respectively. This is comparable to the size of a single feature vector from a ResNet-18 model.



(a) LwF [29] (b) LUCIR [21]

Figure 4.5 – Learned α^k curves on ImageNet-100 for LwF and Lucir across $S = 10$ incremental states. Each curve corresponds to the set of α^k parameters learned after each state, following the color code given above the graphics.

Parameter aggregation. When $R > 1$, a transfer function is needed to apply the parameters learned on multiple reference datasets to a target dataset. We transfer parameters using the averaged α_s^k and β_s^k values, obtained for the set of $\mathcal{D}^r$. In Section 5.1, we evaluate our transfer scheme against an upper-bound oracle which selects the best $\mathcal{D}^r$ in each state.

Dynamic bias correction. Another notable property of our bias correction method is the fact that following Equation (4.3), all previously learned parameters are re-adjusted in each state, instead of keeping them constant. In Figure 4.5 we highlight the advantages of this property by plotting learned α^k correction curves across successive states, for Lucir and LwF trained on ImageNet-100. Optimal bias correction curves need to be progressively adjusted, as bias towards past classes clearly increases across successive states. Again, learned correction parameters seem more stable when learned on top of Lucir, while parameters learned with LwF are less stable.

The proposed approach adds a simple but effective linear layer to calibrate the predictions of backbone class-incremental learning methods. Consequently, it is applicable to any incremental leaning method operating without memory and making use of a classifier layer. In the following section, we test the genericity of the approach by applying it on top of four existing incremental learning methods.

Algorithm 1: Optimizing *adBiC*

input : $\mathcal{A}; \mathcal{D}_s^r \ \forall s \in [\![1, S]\!]$
output : reference parameters $\{\boldsymbol{\theta}_2^r, \ldots, \boldsymbol{\theta}_S^r\}$
$\mathcal{M}_1^r \leftarrow \text{train}(\mathcal{A}; \mathcal{M}_1^r, \mathcal{D}_1^r)$;
for $s = 2 \ldots S$ **do**
  $\mathcal{M}_s^r \leftarrow \text{update}(\mathcal{A}; \mathcal{M}_{s-1}^r, \mathcal{D}_s^r)$;
  $\alpha_s^k \leftarrow 1, \ \beta_s^k \leftarrow 0$ for each $k \in [\![1, s]\!]$;
  foreach $(\mathbf{x}, y) \in \textsc{Validation}(\mathcal{D}^r, s)$ **do**
    $\mathbf{o_s} \leftarrow \mathcal{M}_s^r(\mathbf{x})$;
    for $k = 1 \ldots s$ **do**
      $\mathbf{o_s^k} \leftarrow adBiC(\boldsymbol{o_s^k}) = \alpha_s^k \boldsymbol{o_s^k} + \beta_s^k \cdot \mathbf{1}$;
    end for
    $\mathbf{q_s} \leftarrow \sigma(\mathbf{o_s})$;
    loss $\leftarrow \mathcal{L}(\boldsymbol{q_s}, y)$;
    $\boldsymbol{\theta}_s^r = (\alpha_s^1, \beta_s^1, \ldots, \alpha_s^s, \beta_s^s) \leftarrow \text{optimize(loss)}$;
  end foreach
end for

Algorithm 2: *adBiC* inference

input : $\mathcal{A}; (\alpha_s^k, \beta_s^k) \ \forall s \in [\![1, S]\!], \ k \in [\![1, s]\!]; \ \mathcal{D}_s^t$ for $s \in [\![1, S]\!]; \{\boldsymbol{\theta}_2^r, \ldots, \boldsymbol{\theta}_S^r\}$
output : predictions $\widehat{y}$;
$\mathcal{M}_1^t \leftarrow \text{train}(\mathcal{A}; \mathcal{M}_1^t, \mathcal{D}_1^t)$;
for $s = 2 \ldots S$ **do**
  $\mathcal{M}_s^t \leftarrow \text{update}(\mathcal{A}; \mathcal{M}_{s-1}^t, \mathcal{D}_s^t)$;
  foreach $(\mathbf{x}, y) \in \textsc{Test}(\mathcal{D}^t, s)$ **do**
    $\mathbf{o_s} \leftarrow \mathcal{M}_s^t(\mathbf{x})$;
    for $k = 1 \ldots s$ **do**
      $\mathbf{o_s^k} \leftarrow adBiC(\boldsymbol{o_s^k}; \boldsymbol{\theta}_s^r) = \alpha_s^k \boldsymbol{o_s^k} + \beta_s^k \cdot \mathbf{1}$;
    end for
    $\mathbf{q_s} \leftarrow \sigma(\mathbf{o_s})$;
    $\hat{y} \leftarrow \arg\max_{y \in [\![1, N_s]\!]} (\mathbf{q_{s,y}})$;
  end foreach
end for

Figure 4.6 – Algorithms for the transfer scheme proposed. *adBiC* parameters are learned in each state for reference datasets in Algorithm 1, and then transferred at inference for models trained on target datasets in Algorithm 2. $\mathcal{M}^r$ and $\mathcal{M}^t$ denote incremental models trained on reference and target datasets respectively.

— 5 —
Results

We first describe in Section 5.1 our experimental setup, with details on metrics used, implementation, datasets, and the backbone methods utilized. We then provide a detailed evaluation of our method under various settings in Section 5.1. Finally, in Section 5.3, we propose to evaluate our method under more challenging settings, and we further analyze the effects of bias correction on output scores with our method.

5.1 Experimental setup

5.1.1 Evaluation

We describe here the metrics used to evaluate our strategy, the backbone methods selected on top of which our method is applied, and the additional baselines considered.

Metrics. Various metrics applicable to the class-incremental learning scenario have been proposed in previous works [45, 46]. In this work, we focus on the performance of models on each task, which we measure using the *average incremental accuracy*. First proposed in [45], this metric is the mean of the the accuracy values obtained for each incremental state, and is defined as:

$$A_S = \frac{1}{S-1} \sum_{s=2}^{S} \left(\frac{1}{s} \sum_{k=1}^{s} a_{s,k} \right) \tag{5.1}$$

where S is the total number of states, and $a_{s,k}$ denotes the accuracy of the model learned in state s on task k, defined as:

$$a_{s,k} = \frac{1}{|\mathcal{D}_k|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}_k} \mathbb{1}_{\{y\}} \left(\arg\max_{c\in N_s} \ (\boldsymbol{q}_s \propto \boldsymbol{x})_c \right) \tag{5.2}$$

where $\mathbb{1}$ is the indicator function and $\boldsymbol{q}_s \propto \boldsymbol{x}$ denotes the corrected softmax prediction associated to sample $\boldsymbol{x}$, following Equation (4.4). In each state, the model is thus evaluated on the classes from the current task, alongside classes from all past classes. Note that the performance on the first state $s = 1$ is discarded, as it is not incremental, and does not bring any relevant information regarding the ability of the model to retain performance on previous tasks. However, the performance on the first state will naturally have an impact on the performance on the first task for models in subsequent states.

Backbone CIL methods. Our proposed method is applicable to a wide range of class-incremental learning methods able to operate without a memory. In particular, we focus here on backbone methods applicable to image classification and keeping computational requirements constant during the incremental learning process. As an additional requirement, methods considered should be "architecture agnostic" and applicable to any vision model.

Here, we apply *adBiC* on top of four backbone methods usable for class-incremental learning compliant with these requirements (all previously described in Section 3.2):

- LwF [45] - multi-class formulation of the original method from [29] exploiting knowledge distillation to reduce catastrophic forgetting for past classes, first proposed in [45]. This reformulation is also dubbed as "LwF-Mc" in the literature.

- Lucir [21] - distillation-based approach with additional constraints on class embeddings, and a cosine normalization of the output layer.

- Ft$^+$ [36] - fine-tuning in which past classes weights are not updated to reduce catastrophic forgetting.

- Siw [8] - similar to Ft$^+$, but with the addition of a standardization of class weights in order to improve the comparability of predictions between past and new classes.

Baselines. Additionally to the results obtained when applying *adBiC* on top of the previously mentioned methods, we also provide results obtained with the original *BiC* layer. Following the procedure described in Section 4.2, *BiC* parameters are learned on reference datasets and then transferred to models trained on target datasets during inference. The *BiC* layer is implemented following [55] (see Equation (4.1)), and trained using the same loss function.

We also provide results with an optimal version of *adBiC*, which is obtained via an oracle-based selection of reference parameters. Instead of averaging parameters trained on reference datasets, the best performing reference datasets for each incremental state are selected. More formally, for each state s, the set $\boldsymbol{\theta}_s^* = (\alpha_s^1, \beta_s^1, \ldots, \alpha_s^s, \beta_s^s)$ of optimal *adBiC* parameters to use is defined as:

$$\boldsymbol{\theta}_s^* = \underset{r \in [\![1,R]\!]}{\arg\max}\ \mathbb{O}\left(\mathcal{M}_s^t, \theta_s^r; s\right) \tag{5.3}$$

Where:

- R is the total number of reference datasets.

- θ_s^r is the set of *adBiC* parameters learned on reference dataset $\mathcal{D}^r$ for state s, following Algorithm 1.

- $\mathcal{M}_s^t$ is the model incrementally trained on the target dataset in state s.

And finally, $\mathbb{O}(\mathcal{M}, \theta; s)$ is an oracle function providing the accuracy of model $\mathcal{M}$ on task s, when corrected with *adBiC* parameters θ (following Algorithm 2). This oracle baseline is important, as it indicates the potential supplementary gain obtainable with a parameter selection method more refined than the proposed one.

Finally, we provide results with a training from scratch with all data available at all times (noted *Joint* in the following sections). This method is not incremental, and is an upper bound for all incremental learning strategies.

5.1.2 Implementation details

In this section, we give implementation details of *adBiC* and the tested backbone methods. We provide hyperparameters used and learning strategies, alongside other technical details.

Network architecture

The choice of the backbone network architecture used in incremental image classification tasks varies in the literature. Methods like LwF or Lucir use ResNet [19] models with either 18 or 32 layers, but some methods also use specialized architectures. Changing the backbone network has been shown to influence the performance rankings of incremental learning methods in [36]. In order to propose a unified evaluation of all methods, we use a ResNet-18 backbone in all of our experiments.

Backbone methods

For Lucir [21] and Siw [8], we used the original codes provided by the authors. For LwF, we adapted the multi-class TensorFlow [1] implementation from [45] to incremental learning without memory. For Ft$^+$, we implemented the method by replacing classification weights for each class group by their initial weights learned when classes were encountered for the first time.

For LwF, we use a base learning rate of 1.0 divided by 5 after 20, 30, 40 and 50 epochs. The weight decay is set to 10^{-5} and models are trained for 70 epochs in each state. For Lucir, we mostly use the parameters recommended for Cifar-100 in the original paper [21]. We set λ_{base} to 5. For each state, we train models for 160 epochs. The base learning rate is set to 0.1 and divided by 10 after 80 and 120 epochs. The weight decay is set to $5 \cdot 10^{-4}$ and the momentum to 0.9. Note that since no memory of past classes is available, the margin ranking loss is unusable and thus removed.

Siw and Ft$^+$ are both trained with the same set of hyperparameters. Following [8], models are trained from scratch for 300 epochs in the first non-incremental state, using the SGD optimizer with momentum 0.9. The base learning rate is set to 0.1, and is divided by 10 when the loss plateaus for 60 epochs. The weight decay is set to $5 \cdot 10^{-4}$. For incremental states, the same hyperparameters are used, except for the number of epochs which is reduced to 70 and the lr is divided by 10 when the loss plateaus for 15 epochs. Finally, all methods use the same batch size value of 128.

Adaptive bias correction

Method. The correction of raw output scores is done in the same way for all methods. After the extraction of raw scores and corresponding labels for models learned in each incremental state, batches are fed into a PyTorch [42] module which performs the optimization of *adBiC* parameters, or the transfer of previously learned parameters depending on the setting. Following [36], *BiC* and *adBiC* layers are implemented as pairs of parameters and optimized simply through backpropagation.

Optimization. The parameters α_s^k, β_s^k corresponding to each incremental state s are optimized for 300 epochs, with the Adam [25] optimizer and a starting learning rate of 10^{-3}.
An L2-penalty is added to the loss given in Equation (4.3), with a lambda of $5 \cdot 10^{-3}$ for α parameters and $5 \cdot 10^{-2}$ for β parameters.

5.1.3 Datasets

In this section, we provide some technical details on the datasets selected for our experiments, and the procedure use to construct them.

Reference datasets. The preliminary analysis from Figure 4.4 indicates that bias correction parameters are rather stable across different reference datasets. For our experiments, we make use of multiple reference datasets in order to stabilize the averaged bias correction parameters. Specifically, we use $R = 10$ reference datasets, each including 100 randomly chosen leaf classes from ImageNet [12] with a 500/200 train/validation split per class. All constructed reference datasets have disjoint sets of classes.

Target datasets. We test our method with a total of five target datasets. They were selected to include different types of visual content and thus test the robustness of the parameter transfer. Specifically, we try to select target datasets while varying the following characteristics:

- domain-specificity [1]
- granularity
- semantic level of classes
- proximity to ImageNet

The class samples from the target datasets are split into 500/100 train/test subsets respectively. There is no intersection between classes from the reference datasets and the two target datasets which are sampled from ImageNet (ImageNet-100 and Birds-100).

[1] As an example, methods like Lucir which strongly enforce stability of the learned feature representations will perform better on datasets in which small domain shifts are expected from one task to the next [36, 9].

We briefly describe target datasets hereafter:

- CIFAR-100 [27] - object recognition dataset. It focuses on commonsense classes and is relevant for basic level classification in the sense of [47].
- IMAGENET-100 - subset of ImageNet [12] which includes a hundred randomly selected leaf classes. It is built with the same procedure used for reference datasets and is thus most similar to them. IMAGENET-100 is relevant for fine-grained classification with a diversity of classes.
- BIRDS-100 - dataset built using a hundred bird classes from the ImageNet [12] dataset. It is thus relevant for domain-specific fine-grained classification.
- FOOD-100 - dataset built using a hundred food classes from the Food-101 [10] dataset. It is also a fine-grained and domain-specific dataset, and is relevant as it is independent from ImageNet.

Additionally, we perform experiments on the PLACES-100 dataset built from Places-365 [60], to further investigate the effects of a large domain shift from reference to target datasets on our proposed parameter transfer scheme. We illustrate our dataset transfer scheme in Figure 5.1 below.

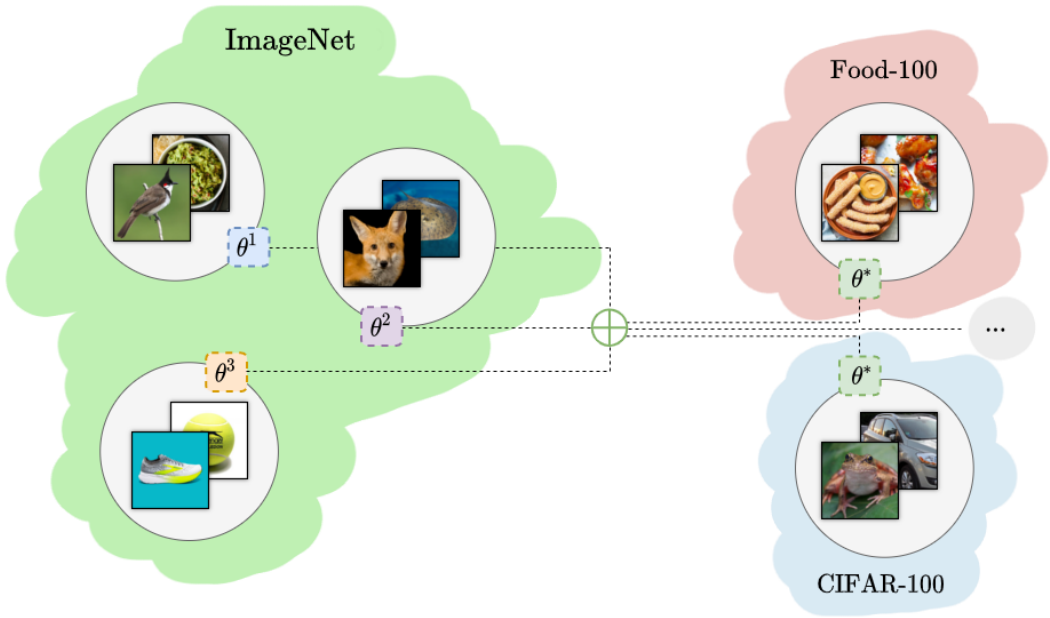


Figure 5.1 – Proposed dataset transfer scheme to evaluate our method. We learn calibration parameters on reference datasets sampled from ImageNet, aggregate them by averaging and transfer them to models learned on FOOD-100, CIFAR-100, PLACES-100 and IMAGENET-100.

5.2 Results and discussion

We provide here the main results of our method on the target datasets presented in Section 5.1.3. In the following tables:

- *Joint* denotes the baseline with full training data available at all times described in Section 5.1.1
- *adBiC* + $\mathbb{O}$ denotes the oracle baseline described in Section 5.1.1

Results are presented for each method without parameter transfer and with *BiC* and *adBiC* transfer.
In Tables 5.1 and 5.2, we provide results for CIFAR-100/IMAGENET-100 and BIRDS-100/FOOD-100. In Table 5.3, partial results on PLACES-100 are also provided. Finally, in Figure 5.2, detailed accuracies for all methods are provided as a plot for $S = 10$ on the CIFAR-100, FOOD-100 and BIRDS-100 datasets, highlighting the gains obtained over the backbone methods after correction. In the Appendix 7, figures for all settings and datasets are also provided.

5.2.1 Main results

Accuracy tables

Method	CIFAR-100			IMAGENET-100		
	$S = 5$	$S = 10$	$S = 20$	$S = 5$	$S = 10$	$S = 20$
LwF	53.0	44.0	29.1	53.8	41.1	29.2
w/ BiC	54.0 + **1.0**	45.5 + **1.5**	30.8 + **1.7**	54.7 + **0.9**	42.5 + **1.4**	31.1 + **1.9**
w/ adBiC	54.3 + **1.3**	**46.4** + **2.4**	**32.3** + **3.2**	55.1 + **1.3**	43.4 + **2.3**	**32.3** + **3.1**
w/ adBiC + $\mathbb{O}$	54.9 + **1.9**	47.3 + **3.3**	32.6 + **3.5**	55.9 + **2.1**	44.2 + **3.1**	33.1 + **3.9**
LUCIR	50.1	33.7	19.5	48.3	30.1	17.7
w/ BiC	52.5 + **2.4**	37.1 + **3.4**	22.4 + **2.9**	54.9 + **6.6**	36.8 + **6.7**	21.8 + **4.1**
w/ adBiC	**54.8** + **4.7**	42.2 + **8.5**	28.4 + **8.9**	**59.0** + **10.7**	**46.1** + **16.0**	27.3 + **9.6**
w/ adBiC + $\mathbb{O}$	55.5 + **5.4**	43.6 + **9.9**	31.2 + **11.7**	59.4 + **11.1**	46.6 + **16.5**	29.7 + **12.0**
SIW	29.9	22.7	14.8	32.6	23.3	15.1
w/ BiC	31.4 + **1.5**	22.8 + **0.1**	14.7 - **0.1**	33.9 + **1.3**	22.6 - **0.7**	13.9 - **1.2**
w/ adBiC	31.7 + **1.8**	24.1 + **1.4**	15.8 + **1.0**	35.1 + **2.5**	24.5 + **1.2**	15.0 - **0.1**
w/ adBiC + $\mathbb{O}$	32.8 + **2.9**	25.0 + **2.3**	16.5 + **1.7**	36.4 + **3.8**	25.7 + **2.4**	16.1 + **1.0**
FT+	28.9	22.6	14.5	31.7	23.2	14.6
w/ BiC	30.7 + **1.8**	22.5 - **0.1**	14.8 + **0.3**	33.0 + **1.3**	21.9 - **1.3**	13.8 - **0.8**
w/ adBiC	31.9 + **3.0**	23.6 + **1.0**	15.0 + **0.5**	34.9 + **3.2**	23.7 + **0.5**	15.7 + **1.1**
w/ adBiC + $\mathbb{O}$	32.5 + **3.6**	24.6 + **2.0**	15.9 + **1.4**	35.7 + **4.0**	24.9 + **1.7**	16.2 + **1.6**
Joint		72.7			75.5	

Table 5.1 – Average top-1 incremental accuracy on CIFAR-100 and IMAGENET-100 using $S = \{5, 10, 20\}$ states, for LWF, LUCIR, SIW and FT$^+$. Gains and losses over the backbone methods are in green/red respectively. Best results for each setting are indicated in bold.

Method	Birds-100			Food-100		
	$S = 5$	$S = 10$	$S = 20$	$S = 5$	$S = 10$	$S = 20$
LwF	53.7	41.8	30.1	42.9	31.8	22.2
w/ BiC	54.6 + **0.9**	43.1 + **1.3**	31.8 + **1.7**	43.4 + **0.5**	32.6 + **0.8**	23.8 + **1.6**
w/ adBiC	55.0 + **1.3**	44.0 + **2.2**	**32.8** + **2.7**	43.5 + **0.6**	33.3 + **1.5**	**24.7** + **2.5**
w/ adBiC + $\mathbb{O}$	55.8 + **2.1**	44.8 + **3.0**	33.3 + **3.2**	44.0 + **1.1**	34.2 + **2.4**	25.3 + **3.1**
LUCIR	50.8	31.4	17.9	44.2	26.4	15.5
w/ BiC	56.0 + **5.2**	37.7 + **6.3**	20.6 + **2.7**	49.9 + **5.7**	31.5 + **5.1**	17.2 + **1.7**
w/ adBiC	**58.5** + **7.7**	**45.4** + **14.0**	27.3 + **9.4**	**52.0** + **7.8**	**37.1** + **10.7**	17.7 + **2.2**
w/ adBiC + $\mathbb{O}$	59.0 + **8.2**	46.0 + **14.6**	28.8 + **10.9**	52.6 + **8.4**	38.2 + **11.8**	21.0 + **5.5**
SIW	30.6	23.2	14.9	29.4	21.6	14.1
w/ BiC	32.8 + **2.2**	22.7 - **0.5**	12.8 - **2.1**	29.1 - **0.3**	20.3 - **1.3**	12.1 - **2.0**
w/ adBiC	33.0 + **2.4**	25.2 + **2.0**	15.3 + **0.4**	30.9 + **1.5**	21.3 - **0.3**	14.5 + **0.4**
w/ adBiC + $\mathbb{O}$	34.4 + **3.8**	26.2 + **3.0**	16.3 + **1.4**	31.5 + **2.1**	22.6 + **1.0**	15.1 + **1.0**
FT+	29.7	23.3	13.5	28.7	21.1	13.3
w/ BiC	32.3 + **2.6**	22.5 - **0.8**	12.4 - **1.1**	28.6 - **0.1**	20.6 - **0.5**	11.8 - **1.5**
w/ adBiC	34.0 + **4.3**	25.0 + **1.7**	14.2 + **0.7**	30.8 + **2.1**	22.2 + **1.1**	14.2 + **0.9**
w/ adBiC + $\mathbb{O}$	34.5 + **4.8**	25.7 + **2.4**	15.4 + **1.9**	31.3 + **2.6**	22.7 + **1.6**	14.5 + **1.2**
Joint		80.9			71.03	

Table 5.2 – Average top-1 incremental accuracy on Birds-100 and Food-100 using $S = \{5, 10, 20\}$ states, for LwF, Lucir, Siw and Ft$^+$. Gains and losses over the backbone methods are in green/red respectively. Best results for each setting are indicated in bold.

Method	Places-100		
	$S = 5$	$S = 10$	$S = 20$
LwF	43.3	35.1	25.9
w/ BiC	43.9 + **0.6**	36.1 + **1.0**	27.6 + **1.7**
w/ adBiC	**44.2** + **0.9**	**36.6** + **1.5**	**28.6** + **2.7**
w/ adBiC + $\mathbb{O}$	44.6 + **1.3**	37.5 + **2.4**	29.3 + **3.4**
LUCIR	40.5	26.0	16.0
w/ BiC	42.6 + **2.1**	29.9 + **3.9**	18.0 + **2.0**
w/ adBiC	42.8 + **2.3**	35.4 + **9.4**	23.3 + **7.3**
w/ adBiC + $\mathbb{O}$	43.7 + **3.2**	36.5 + **10.5**	24.9 + **8.9**
SIW	27.3	20.6	14.0
w/ BiC	26.7 - **0.6**	20.6 + **0.0**	12.0 - **2.0**
w/ adBiC	28.8 + **1.5**	21.2 + **0.6**	13.1 - **0.9**
w/ adBiC + $\mathbb{O}$	29.0 + **1.7**	22.1 + **1.5**	14.1 + **0.1**
FT+	26.9	20.8	12.1
w/ BiC	25.9 - **1.0**	19.8 - **1.0**	10.9 - **1.2**
w/ adBiC	27.3 + **0.4**	19.7 - **1.1**	13.2 + **1.1**
w/ adBiC + $\mathbb{O}$	28.4 + **1.5**	21.2 + **0.4**	14.0 + **1.9**

Table 5.3 – Average top-1 incremental accuracy on Places-100 using $S = \{5, 10, 20\}$ states, for LwF, Lucir, Siw and Ft$^+$. Gains and losses over the backbone methods are in green/red respectively. Best results for each setting are indicated in bold.

Accuracy plots

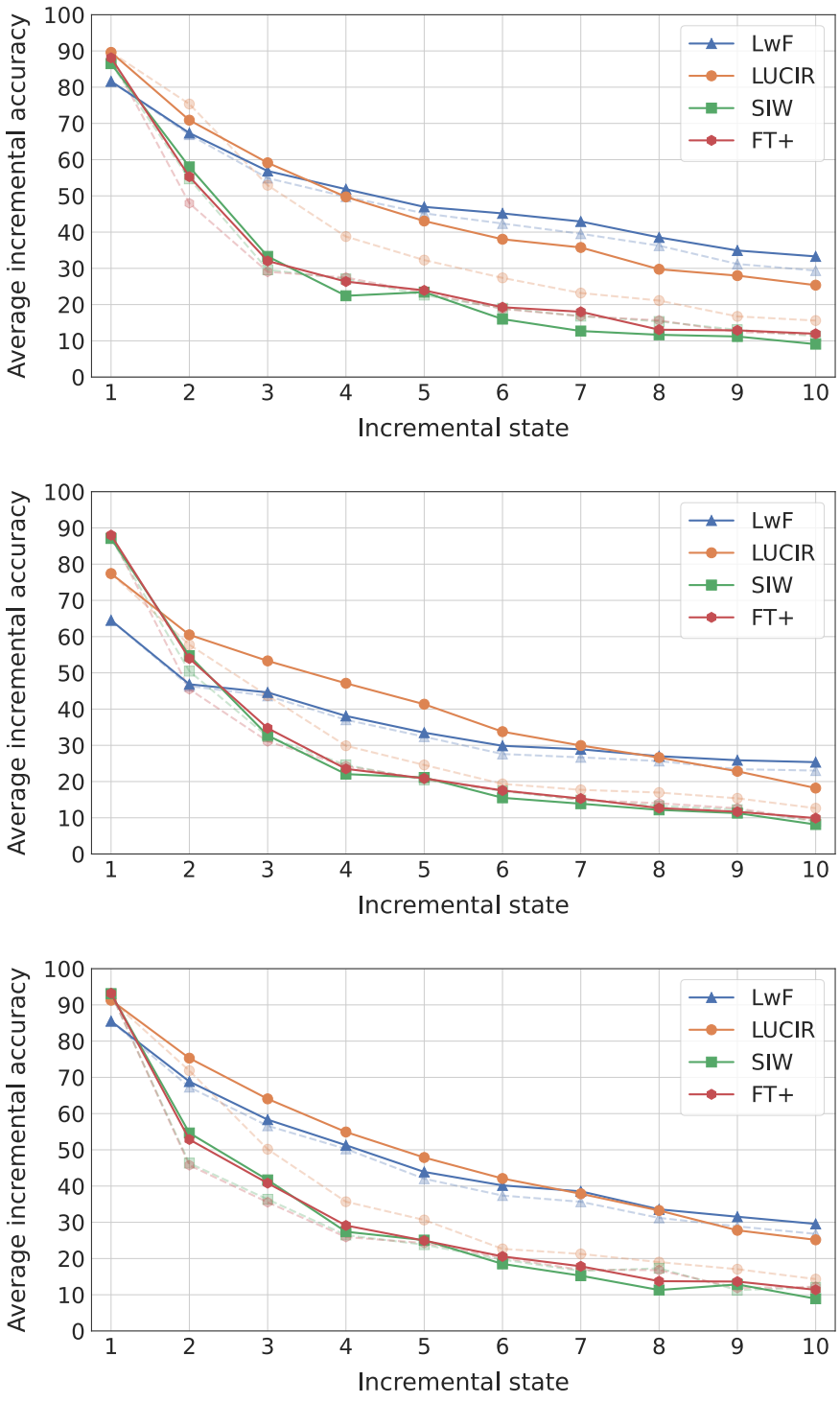


Figure 5.2 – Average top-1 accuracies in each state on CIFAR-100 (top), FOOD-100 (middle) and BIRDS-100 (bottom), with all backbone methods after *adBiC* correction, for $S = 10$. Accuracies without correction of the corresponding methods are provided in dotted lines (same colors). Plots for all datasets are also provided in the appendix.

5.2.2 Discussion

General results. Overall, results show that our method improves the performance of baseline methods for all but two of the configurations evaluated. The best overall performance before bias correction is obtained with LwF. This result confirms the conclusions of [8, 36] regarding the strong performance of LwF in class IL without memory for medium-scale datasets. With *adBiC*, Lucir performs generally better than LwF for $S = 5$ and $S = 10$, while LwF remains stronger with $S = 20$ states.
Results are particularly strong for Lucir, a method for which *adBiC* brings consistent gains in most configurations, with up to 16 accuracy points over the backbone accuracy. Tables 5.1, 5.2 and 5.3 show that *adBiC* also improves the results of LwF in all configurations, albeit to a lesser extent compared to Lucir.

Improvements for LwF are larger for $S = 20$ states, which is the most challenging configuration since the model is more prone to forgetting. Ft$^+$ [36] and Siw [8] remove the distillation component for the class-incremental training process and exploit the weights of past classes learned in their initial state.
adBiC improves results for these two methods in almost all configurations. However, their global performance is significantly lower than that of LwF and Lucir, the two methods which make use of distillation. This result confirms the finding from [8] regarding the usefulness of the distillation term exploited by LwF and Lucir to stabilize IL training for medium scale datasets.

Effectiveness of *adBiC*. Results from Tables 5.1 highlight the effectiveness of *adBiC* compared to *BiC*. *adBiC* provides better gains in all tested configurations, with the most important gain over *BiC* obtained for Lucir. It is also worth noting that *adBiC* improves results for Siw and Ft$^+$ in most configurations, while the corresponding results of *BiC* are mixed. The comparison of *adBiC* and *BiC* validates our hypothesis that a finer-grained modeling of forgetting for past states is a better way to perform calibration.

Oracle upper bound. We also compare *adBiC*, which uses averaged α and β parameters, with an oracle selection of parameters (denoted as $+\ \mathbb{O}$ in Tables 5.1, 5.2 and 5.3). The performance of *adBiC* is close to this upper bound for all tested methods, with a difference of less than one accuracy point in the majority of settings. This indicates that averaging is an effective way to aggregate parameters learned from reference datasets. However, investigating more refined ways to select reference parameters for a given target dataset may further improve performance.

Datasets and shift. The comparison of target datasets shows that the gain brought by *adBiC* is largest for ImageNet-100, followed by Birds-100, Cifar-100 and Food-100. This is intuitive as ImageNet-100 has the closest distribution to that of reference datasets. Birds-100 is extracted from ImageNet and, while topically different from reference datasets, was created using similar guidelines. The consistent improvements obtained with Cifar-100 and Food-100, two datasets independent from ImageNet, shows that the proposed transfer method is robust to data distribution changes. Similarly, results on Places-100 for LwF and Lucir are comparable to those obtained on other target datasets, despite the domain shift from ImageNet.

Effects of the number of states. Except for LwF, *adBiC* gains are larger for $S = \{5, 10\}$ compared to $S = 20$. This result is consistent with past findings reported for bias correction methods [36, 55]. It is mainly explained by the fact that the size of validation sets needed to optimize *adBiC* parameters is smaller and thus less representative for larger values of S. A larger number of states leads to a higher degree of forgetting. This makes the IL training process more challenging and also has a negative effect on the usefulness of the bias correction layer.

Finally, the performance gaps between incrementally learned models and the *Joint* upper bound are still wide, particularly for larger values of S. This indicates that class-incremental learning without memory still remains an open challenge.

5.3 Further analysis

5.3.1 Robustness experiments

In this section, we complement the results presented in Section 5.2 with two experiments which further evaluate the robustness of *adBiC*. In particular, we test the effectiveness of our parameter transfer scheme in cases where reference and target models are trained with different volumes of data, and we investigate the effects of the number of reference datasets.

Dataset sizes. Results in Section 5.2.1 indicate that parameter transfer is generally stronger with target datasets that are visually similar to reference datasets. Besides similarity, we now investigate the effects of training the reference and target models with different dataset scales. Intuitively, varying data sizes should affect the amplitude of bias, and thus negatively impact the transferability of learned parameters from reference to target models.

We verify this intuition by reproducing the experiments of Section 5.2, with half of the training images for target datasets compared to reference datasets. Target datasets are thus now trained with 250 training images per class, while reference datasets still keep 500 training images per class.
Results for these experiments are presented in Tables 5.4, 5.5 and 5.6. Overall, the transfer of calibration parameters consistently brings gains in accuracy for LwF and Lucir. Results are more mixed for Siw and Ft$^+$, although gains are still observable.

These results show that our approach remains strong when reference and target datasets differ in size, although this difference clearly impacts the transferability of calibration parameters. Maintaining a low difference in dataset sizes is thus preferable in order to keep the transfer effective.

Number of reference datasets. As described in the method we proposed in Section 4, we propose to aggregate parameters learned on multiple reference datasets simply by averaging them. We propose here to asses the robustness of our method with respect to R, the number of available reference datasets.

We modify this variable from $R = 1$ to $R = 9$. For each R value, we perform 10 random samplings of the set of reference datasets to be used, and report obtained standard deviations on the resulting accuracies. We provide results for Lucir on the Food-100 dataset, which exhibits the largest domain shift from ImageNet out of the selected datasets, in Table 5.7. Additional results for all methods on the Cifar-100 dataset are provided Appendix 7.

Overall, most of the accuracy gains are obtained after a single dataset is used, although using multiple reference datasets does help stabilizing results. This confirms that parameter transfer is effective with a small number of reference datasets.

Method	CIFAR-100 (*halved*)			IMAGENET-100 (*halved*)		
	$S = 5$	$S = 10$	$S = 20$	$S = 5$	$S = 10$	$S = 20$
LwF	41.3	33.3	23.3	45.6	33.5	23.8
w/ adBiC	42.1 + **0.8**	34.8 + **1.5**	25.0 + **1.7**	46.7 + **1.1**	35.3 + **1.8**	25.6 + **1.8**
LUCIR	43.5	27.8	16.6	42.9	27.6	17.0
w/ adBiC	48.3 + **4.8**	38.5 + **10.7**	25.3 + **8.7**	54.1 + **11.2**	42.4 + **14.8**	23.2 + **6.2**
SIW	31.7	21.6	13.7	32.1	22.7	14.4
w/ adBiC	33.7 + **2.0**	22.5 + **0.9**	14.0 + **0.3**	35.0 + **2.9**	22.6 - **0.1**	12.2 - **2.2**
FT+	30.4	21.5	12.9	31.2	22.2	12.0
w/ adBiC	32.0 + **1.6**	21.4 - **0.1**	13.4 + **0.5**	34.8 + **3.6**	21.2 - **1.0**	13.7 + **1.7**

Table 5.4 – Average top-1 incremental accuracy on CIFAR-100 and IMAGENET-100 with half of the training images compared to reference datasets, using $S = \{5, 10, 20\}$ states, for LWF, LUCIR, SIW and FT$^+$. Gains and losses over the backbone methods are in green/red respectively.

Method	BIRDS-100 (*halved*)			FOOD-100 (*halved*)		
	$S = 5$	$S = 10$	$S = 20$	$S = 5$	$S = 10$	$S = 20$
LwF	44.6	34.0	23.2	29.5	23.3	17.3
w/ adBiC	45.5 + **0.9**	35.4 + **1.4**	25.2 + **2.0**	29.9 + **0.4**	24.3 + **1.0**	18.7 + **1.4**
LUCIR	45.2	27.8	16.0	37.9	22.7	13.9
w/ adBiC	52.8 + **7.6**	40.9 + **13.1**	25.6 + **9.6**	45.7 + **7.8**	32.6 + **9.9**	19.8 + **5.9**
SIW	29.7	22.8	14.1	28.4	18.7	13.5
w/ adBiC	32.1 + **2.4**	23.7 + **0.9**	13.5 - **0.6**	29.9 + **1.5**	16.9 - **1.8**	13.3 - **0.2**
FT+	29.2	22.8	12.2	27.4	18.2	11.6
w/ adBiC	31.9 + **2.7**	23.0 + **0.2**	13.6 + **1.4**	28.8 + **1.4**	16.2 - **2.0**	12.2 + **0.6**

Table 5.5 – Average top-1 incremental accuracy on BIRDS-100 and FOOD-100 with half of the training images compared to reference datasets, using $S = \{5, 10, 20\}$ states, for LWF, LUCIR, SIW and FT$^+$. Gains and losses over the backbone methods are in green/red respectively.

Method	PLACES-100 (*halved*)		
	$S = 5$	$S = 10$	$S = 20$
LwF	35.4	27.7	21.5
w/ adBiC	35.9 + **0.5**	28.5 + **0.8**	**23.6** + **2.1**
LUCIR	35.5	23.2	14.7
w/ adBiC	**40.5** + **5.0**	**33.6** + **10.4**	22.3 + **7.6**
SIW	27.2	19.6	14.8
w/ adBiC	28.5 + **1.3**	19.3 - **0.3**	14.3 - **0.5**
FT+	26.1	19.9	12.4
w/ adBiC	25.6 - **0.5**	17.2 - **2.7**	13.5 + **1.1**

Table 5.6 – Average top-1 incremental accuracy on PLACES-100 with half of the training images compared to reference datasets, using $S = \{5, 10, 20\}$ states, for LWF and LUCIR. Gains and losses over the backbone methods are in green/red respectively.

$S = 5$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	44.19	51.9 ± 0.4	52.0 ± 0.2	52.1 ± 0.2	52.0 ± 0.1	52.1 ± 0.1	52.0 ± 0.1	52.0 ± 0.1	52.0 ± 0.1	52.0 ± 0.1	52.0
$S = 10$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	26.44	36.7 ± 0.7	36.9 ± 0.4	37.2 ± 0.4	37.2 ± 0.3	37.1 ± 0.2	37.0 ± 0.2	37.0 ± 0.1	37.1 ± 0.0	37.1 ± 0.1	37.1
$S = 20$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	15.47	17.6 ± 1.2	17.5 ± 0.7	17.6 ± 0.7	17.8 ± 0.4	17.5 ± 0.3	17.7 ± 0.4	17.8 ± 0.3	17.6 ± 0.2	17.7 ± 0.1	17.7

Table 5.7 – Average top-1 incremental accuracy of *adBiC*-corrected models trained incrementally on FOOD-100 with LUCIR, for $S = \{5, 10, 20\}$ states, while varying the number R of reference datasets. For $R \leq 9$, results are averaged across 10 random samplings of the reference datasets (hence the std values). *Raw* is the accuracy of LUCIR without bias correction.

5.3.2 Additional observations

In this section, we further investigate the effects of adaptive bias correction on output scores, and provide empirical evidence to support its effectiveness. We also provide evidence of the relationship between representation drift and maximum task performance. Finally, we compare the maximum performance gains obtainable with a hypothetical bias correction method able to remove all inter-task interference, to our method.

State-wise accuracies

By rescaling output scores for classes learned in specific states, our proposed bias correction method enables a fairer treatment of past classes. Here, we investigate the effects of bias correction by parameter transfer on the accuracy of the models for recent and older tasks.

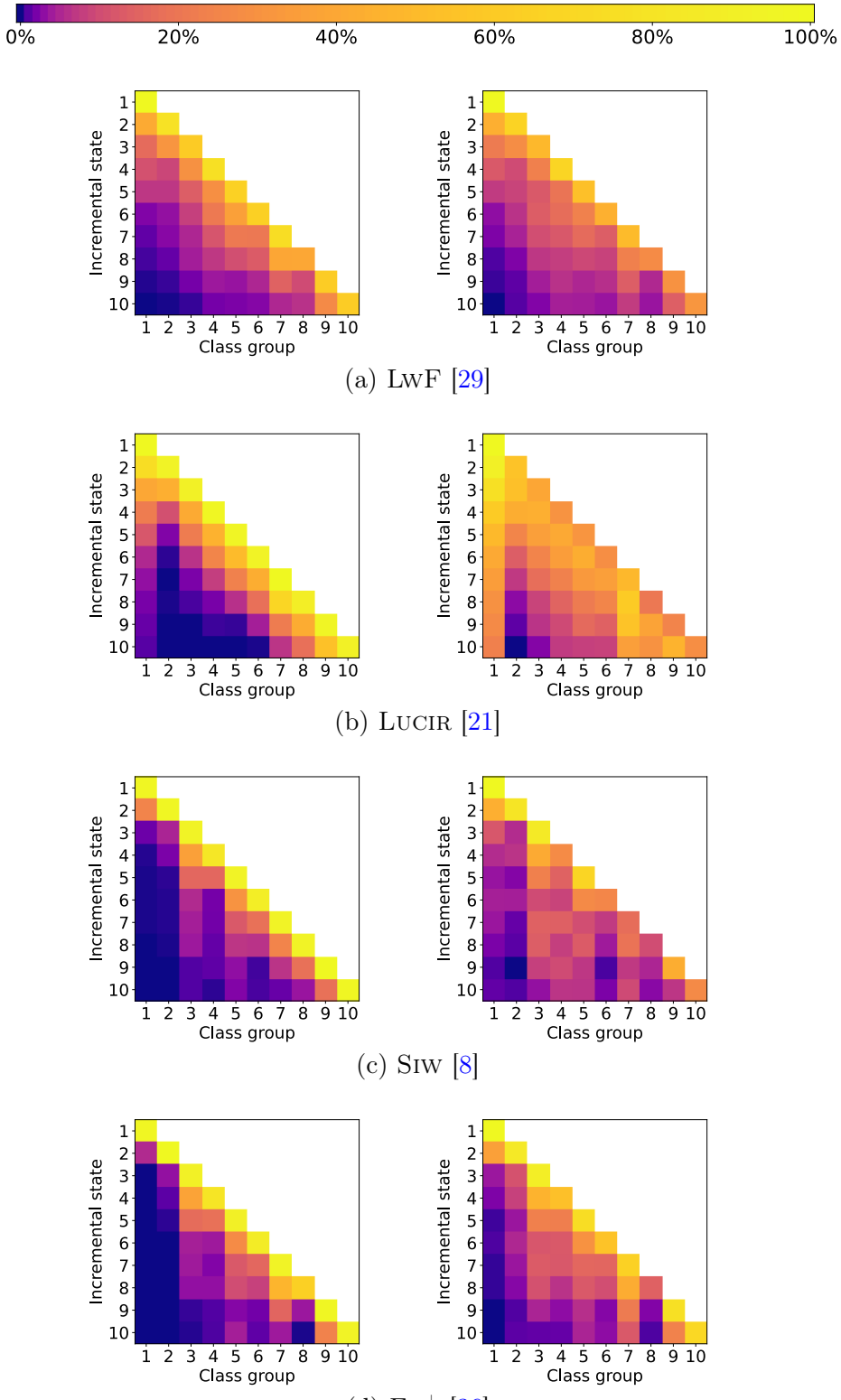
In Figure 5.3, we illustrate the effects of *adBiC* on state-wise accuracies, for all backbone IL methods evaluated in this work. Each row represents an incremental state and each square the accuracy on a group of classes first learned in a specific state. In the first state, represented by the first rows of the matrices, models are only evaluated on the first class group. In the second state, represented by the second rows, models are evaluated on the first two class groups, etc.

Before adaptive correction (left), all methods perform strongly on the last group of classes learned (represented by the diagonals). Remarkably, LUCIR maintains a stronger performance on the first task learned (leftmost column) compared to the other methods. As LUCIR enforces the similarity of learned feature representations across states (see Equation (3.2)), this result is intuitive. Methods that have a higher plasticity like SIW and FT$^+$ do not exhibit this characteristic.

Their performance is generally poorer for past classes (under the diagonals). Overall, the rate of forgetting remains constant across class groups (except in the first class group for LUCIR). With LWF and LUCIR, three to four incremental steps are sufficient for the accuracy on a class group to drop below 20%. On SIW and FT$^+$, the rate of forgetting is predictably faster.

After correction (right), all methods perform better on past class groups resulting in a higher overall performance. Notably, the effects of bias correction varies greatly across class groups. For the first class group learned with LUCIR, bias correction is highly effective - which hints again towards the less-forget constraint favouring the first class group. For SIW and FT$^+$, bias correction on the first class group does not favor the first class groups. This result is expected, as these methods do not enforce stability of the feature extractor.

Finally, and for all methods, the increase in the accuracy on past classes (under the diagonals) is accompanied with a trade-off in the accuracy of models on the last class group (diagonal), which is also an anticipated effect of bias correction.



(a) LwF [29]

(b) Lucir [21]

(c) Siw [8]

(d) Ft^{+} [36]

Figure 5.3 – Accuracies per incremental state for each class group, for models trained with all methods considered on Cifar-100 for $S = 10$ states, before (left) and after (right) *adBiC* correction.

Representation drift and task performance

We are interested here in the drift of feature representations and its relationship with the performance of incrementally learned model on specific tasks.

Evaluating drift. For a specific class $c \in P_k$, we propose to evaluate the average feature drift from state k to state s as the drift of its centroid:

$$\left\| \sum_{\boldsymbol{x} \in \mathcal{C}_{k,c}} \phi_s(\boldsymbol{x}) - \sum_{\boldsymbol{x} \in \mathcal{C}_{k,c}} \phi_k(\boldsymbol{x}) \right\|_F , \; k \leq s \tag{5.4}$$

where $\mathcal{C}_{k,c}$ denotes the set of samples from $\mathcal{D}_k$ belonging to class c:

$$\mathcal{C}_{k,c} = \left\{ \boldsymbol{x} \mid (\boldsymbol{x}, y) \in \mathcal{D}_k \; ; \; y = c \right\} \tag{5.5}$$

We then evaluate the average feature drift of task k to state s as:

$$\psi_k(s) = \sum_{c \in P_k} \frac{1}{|P_k| \cdot |\mathcal{C}_{k,c}|} \left\| \sum_{\boldsymbol{x} \in \mathcal{C}_{k,c}} \phi_s(\boldsymbol{x}) - \sum_{\boldsymbol{x} \in \mathcal{C}_{k,c}} \phi_k(\boldsymbol{x}) \right\|_F , \; k \leq s \tag{5.6}$$

Note that we do not cumulate the average drift between every pair of tasks, but only consider the average feature vector drift between the state in which the task was first learned and the current state.

Maximum task-wise accuracy. In order to estimate the effects of task representation drift, we compare it to the maximum task-wise accuracy obtainable with a bias-corrected incremental model. We define this metric for task k in state s as the accuracy of raw predictions truncated to the first $|N_k|$ outputs, that is:

$$\overline{a_{s,k}} = \frac{1}{|\mathcal{D}_k|} \sum_{(\boldsymbol{x},y) \in \mathcal{D}_k} \mathbb{1}_{\{y\}} \left(\underset{c \; \in N_k}{\arg\max} \; (f_s(\boldsymbol{x}))_c \right) \tag{5.7}$$

with $\mathbb{1}$ the indicator function and f_s the prediction function associated to the model learned in state s as defined in Equation (2.1). Note that this task accuracy is a clear upper bound of what is achievable in incremental learning, as interference with future tasks is completely removed.

Results. In Figure 5.4, we plot the results for a model trained with LUCIR across $S = 20$ states. We compare the normalized feature drift to the minimum task-wise error of the first and tenth tasks. While the error on each task when newly learned is less than 10% on both tasks, it rises to up to 60% error for the first task. Since we remove newer classes activations from output vectors when evaluating maximum task performance, predictions for the considered tasks are not affected by task-recency bias. For both tasks, feature drift clearly negatively correlates with maximum performance for the two tasks considered, which highlights its effect on task performance.

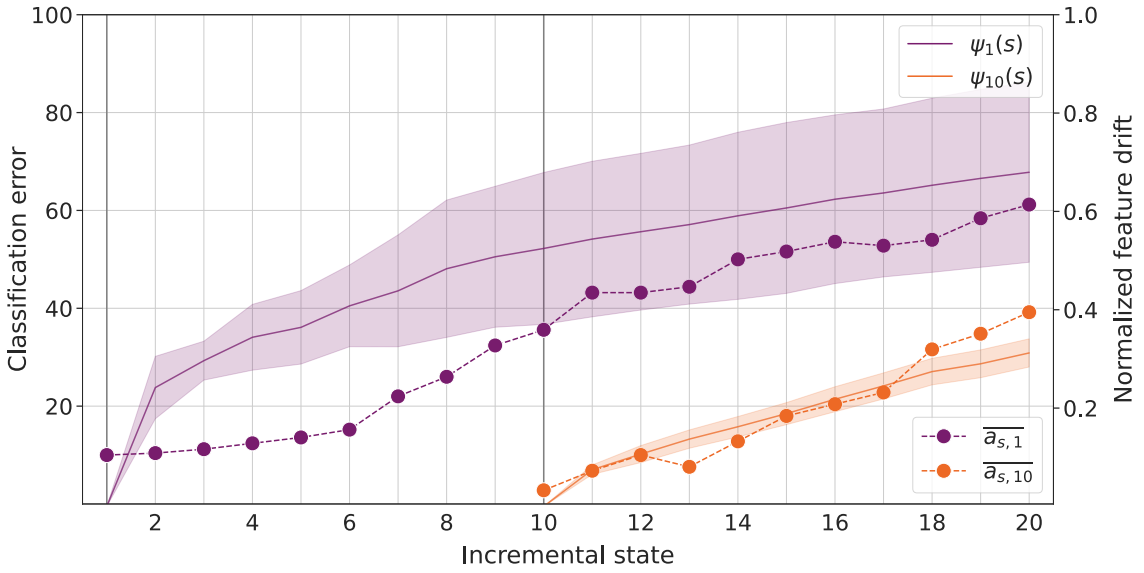


Figure 5.4 – Normalized feature drift (full line, with standard deviation) and minimum task-wise errors (dashed lines) for tasks 1 and 10 across $S = 20$ states, for an incremental model trained with LUCIR on IMAGENET-100. The average feature drift for a given task is clearly correlated with maximum task performance.

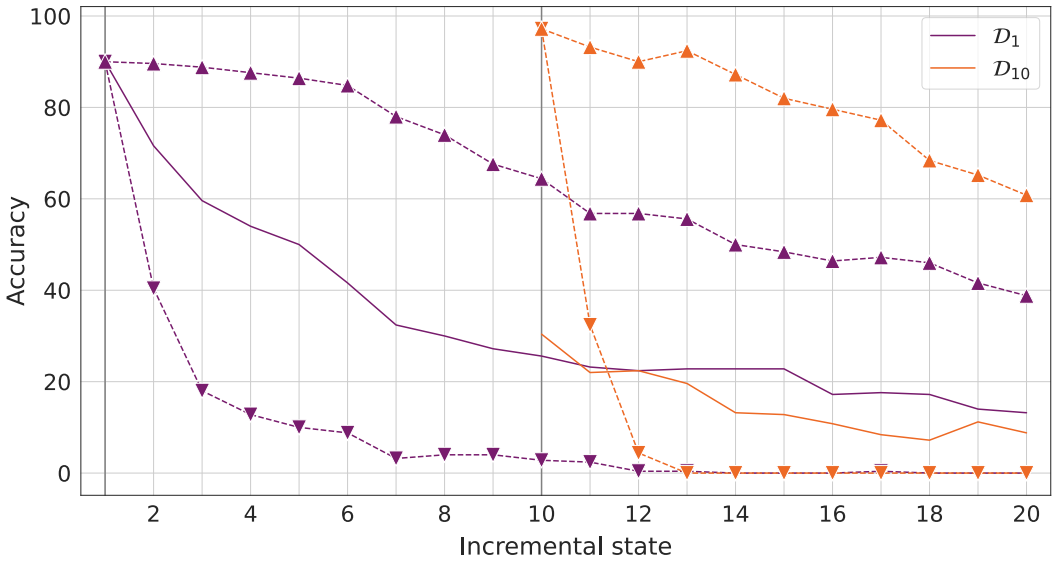


Figure 5.5 – Maximum task-wise accuracy (dashed lines: ▲), task-wise accuracy (dashed lines: ▼) and task-wise accuracy after correction (full lines) for tasks 1 and 10 across $S = 20$ states, with an incremental model trained with LUCIR on IMAGENET-100. While bias correction via *adBiC* partially compensates for the interference in activations of more recent classes, a wide margin of improvement is still possible.

Maximum task-wise accuracy and bias correction

We compare here the maximum task-wise accuracy as defined in Equation (5.7) to task-wise accuracy (Equation (5.2)) and task-wise accuracy after bias correction via *adBiC*. We plot the results for LUCIR trained on IMAGENET-100 across $S = 20$ states in Figure 5.5, for the first and tenth task. For the first task and in the first state, the three metrics are equal since bias correction has no effect when a single task is learned, and there is no interference with future classes. A large gap in accuracy exists between task-wise accuracy with (▼) and without interference of future classes (▲). This gap is caused by task-recency bias, but is also due to inter-task confusion as optimal bias correction (full line) only partially bridges this gap.

These results indicate that while bias correction is an integral cause of catastrophic forgetting in class-incremental learning, compensating for bias is not sufficient to maximally recover task performance. These observations corroborate previous empirical studies such as [49]. Furthermore, even if maximum task-wise performance is fully recovered after bias correction, maximum task-wise accuracy still clearly highlights task forgetting as an effect of representation drift, as previously highlighted in Figure 5.4.

Correction curve sensitivity

We investigate here if replacing learned bias correction curves by a simple model with less parameters would yield similar (or better) accuracy. In Figure 5.6, we compare accuracies obtained when replacing learned alpha parameters for $S = 20$ states by linear and exponential curve fittings.

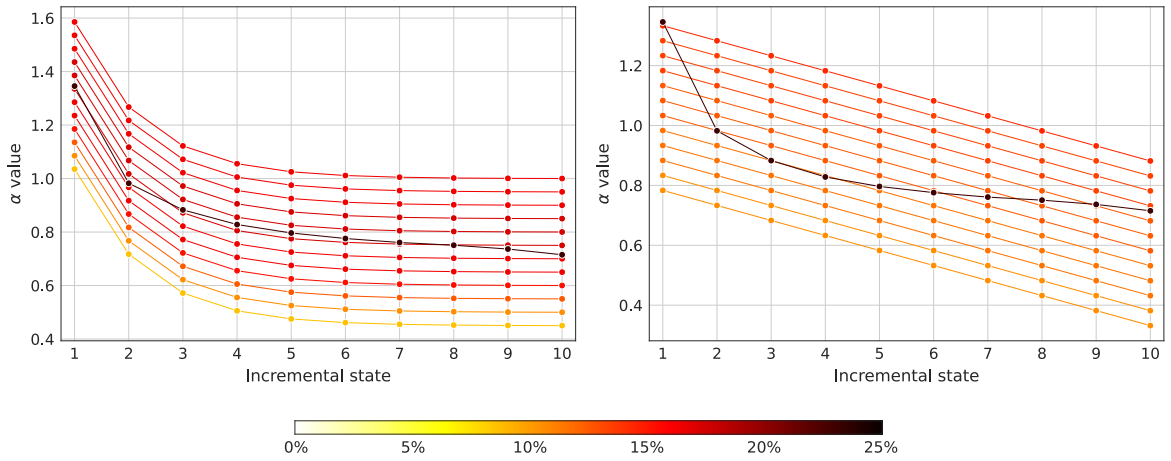


Figure 5.6 – Comparing accuracies obtained when replacing learned alpha parameters for $S = 20$ states by linear and exponential curve fittings, for an incremental model learned with LUCIR on IMAGENET-100. The original alpha curve (in black) yields an accuracy five points higher than the best linear or exponential approximation, in state $S = 20$.

On average, using learned parameters yields an accuracy at least five percents higher than the best exponential/linear alternative, which highlights the importance of learning bias calibration parameters.

— 6 —
Conclusion

In this work, we introduced a method enabling the use of bias correction methods for class-incremental learning without memory, a challenging scenario highly subject to catastrophic forgetting. The proposed method transfers bias correction parameters learned offline from reference datasets toward target datasets. Since reference dataset training is done offline, a validation memory which includes exemplars from all incremental states can be exploited to optimize the proposed correction layer. Our evaluation provides comprehensive empirical support for the transferability of bias correction parameters.

Performance of the state-of-the-art backbone methods employed is improved for almost all configurations tested, with gains up to 16 top-1 accuracy points. Robustness evaluation shows that parameter transfer is efficient when only a small number of reference datasets is used for transfer. It is also usable when the number of training images per class in target datasets is different from that of available reference datasets. These last two findings are important in practice, since the same reference datasets can be exploited in different incremental configurations.

A second contribution relates to the modeling of the degree of forgetting associated to past states. While task-recency bias was already acknowledged [36], no difference was made between past classes learned in different incremental states [55]. This is in part due to validation memory constraints which appear when the bias correction layer is optimized during the incremental process. Such constraints are reduced here since reference datasets training is done offline and a refined definition of the bias correction layer with specific parameters for each past state becomes possible. The comparison of the standard and of the proposed definition of the bias correction layer is favorable to the latter.

The reported results encourage us to pursue the work presented here. First, parameter transfer is done using average values of parameters learned on reference datasets: a finer-grained transfer method could be devised to get closer to the oracle results reported in Section 5.1. Furthermore, we propose here to transfer bias correction parameters from source to target models. Our transfer scheme could be extended to better predict representation drift [57] or to facilitate inter-task separation in a memoryless setting.

— 7 —
Appendix

In this appendix, we provide:

- Detailed accuracy plots for incremental models, before and after correction, for $S = \{5,\ 10,\ 20\}$ for:
  - CIFAR-100, in Figure 7.1
  - IMAGENET-100, in Figure 7.2
  - BIRDS-100, in Figure 7.3
  - FOOD-100, in Figure 7.4
- Additional results when varying the number of reference datasets, in Table 7.1.

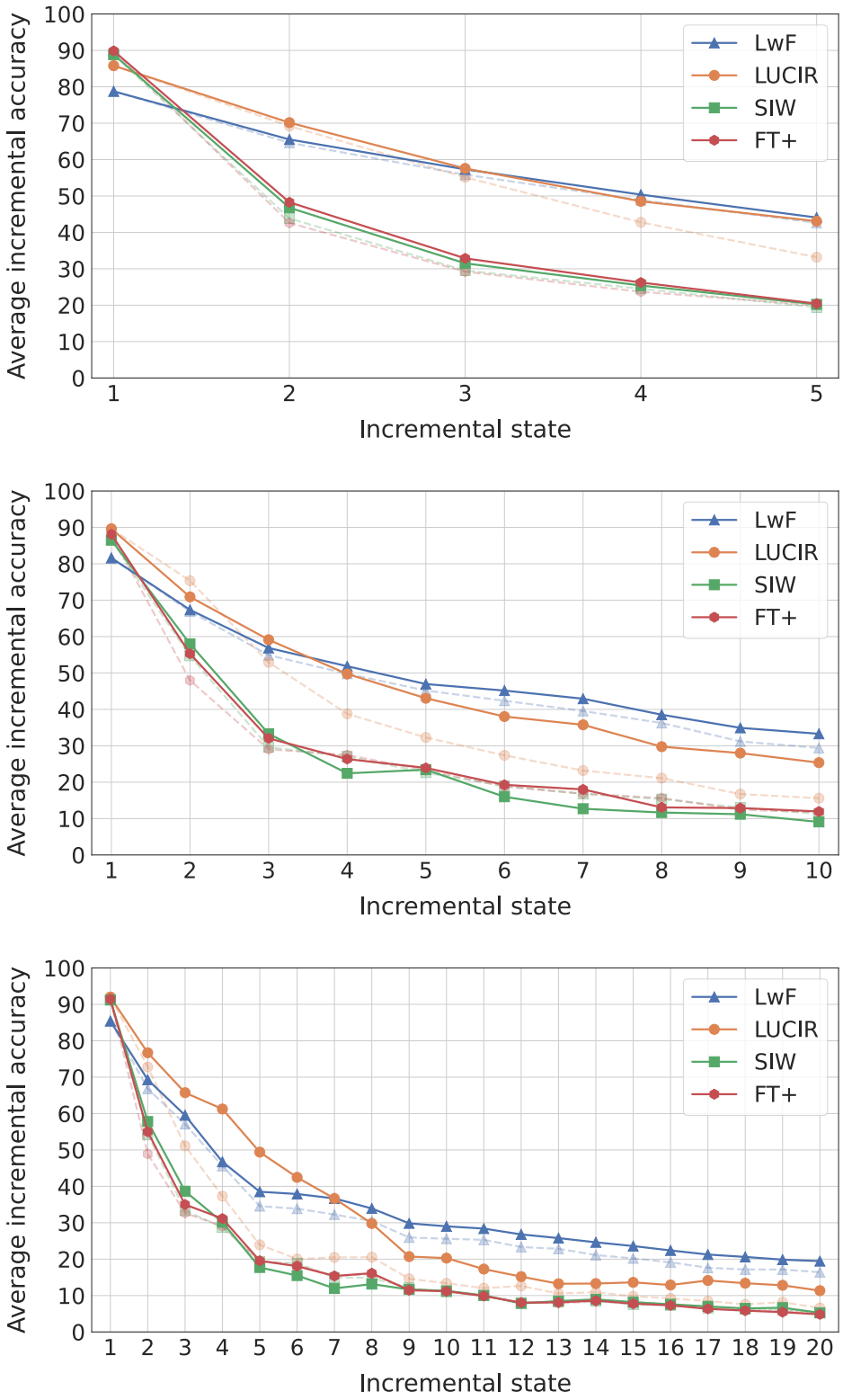


Figure 7.1 – Average top-1 accuracies in each state on CIFAR-100 with all backbone methods after *adBiC* correction, for $S = 5$ (top), $S = 10$ (middle) and $S = 20$ (bottom) states. The accuracies without correction of the corresponding methods are provided in dotted lines (same colors).

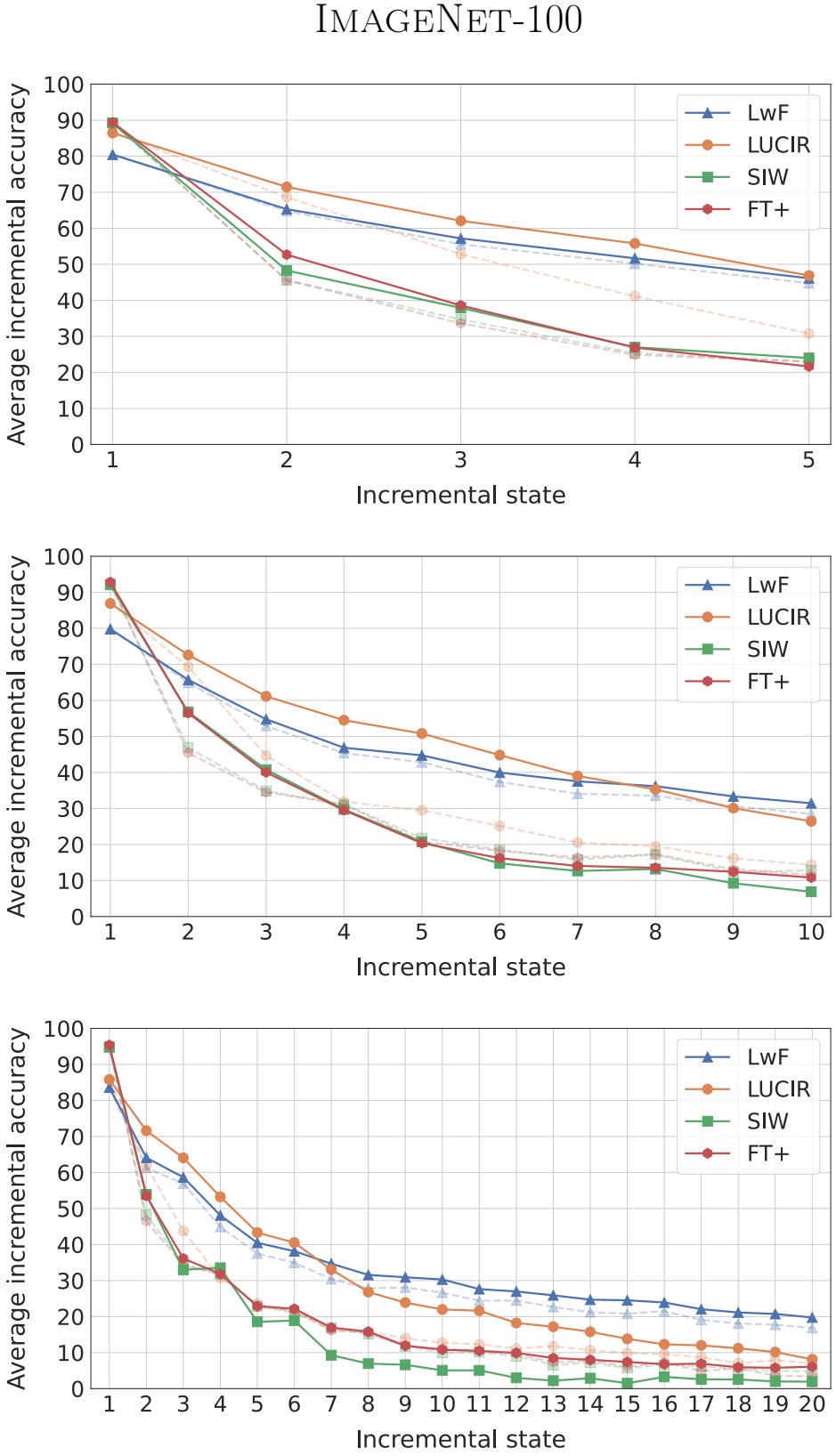


Figure 7.2 – Average top-1 accuracies in each state on ImageNet-100 with all backbone methods, for $S = 5$ (top), $S = 10$ (middle) and $S = 20$ (bottom) states. The accuracies without correction of the corresponding methods are provided in dotted lines (same colors).

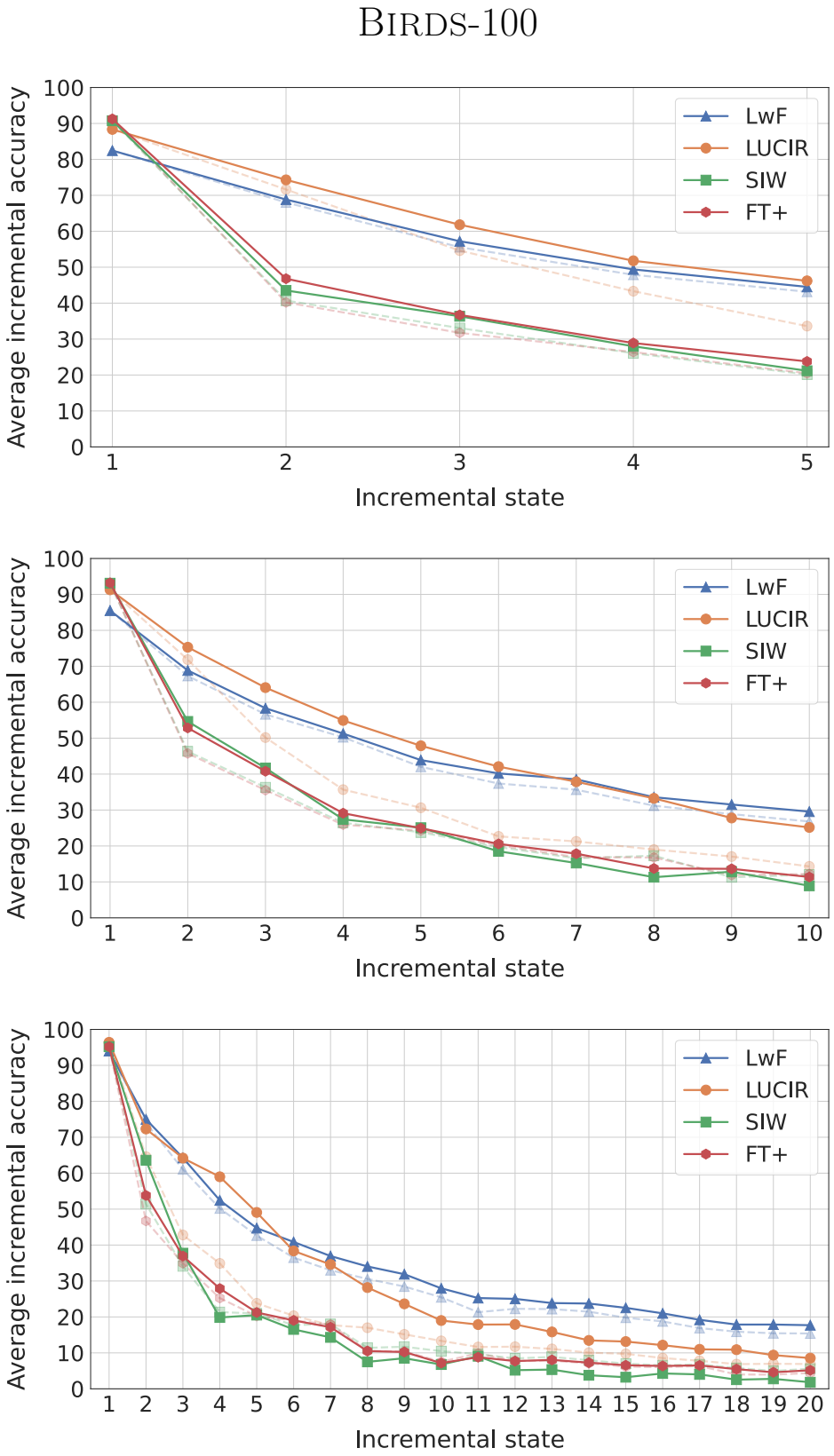


Figure 7.3 – Average top-1 accuracies in each state on BIRDS-100 with all backbone methods, for $S = 5$ (top), $S = 10$ (middle) and $S = 20$ (bottom) states. The accuracies without correction of the corresponding methods are provided in dotted lines (same colors).

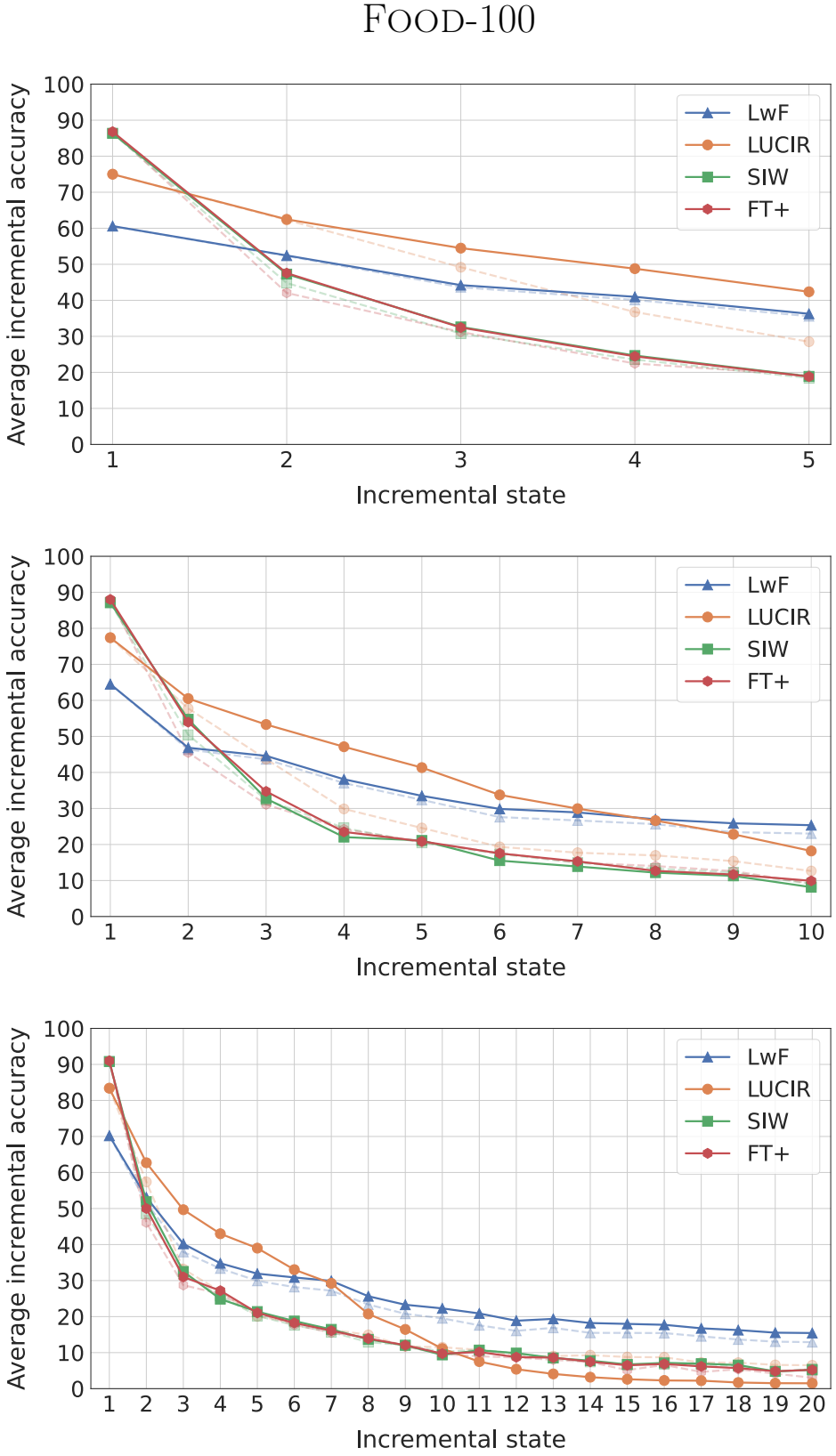


Figure 7.4 – Average top-1 accuracies in each state on FOOD-100 with all backbone methods, for $S = 5$ (top), $S = 10$ (middle) and $S = 20$ (bottom) states. The accuracies without correction of the corresponding methods are provided in dotted lines (same colors).

$S = 5$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	53.0	54.3 ± 0.2	54.3 ± 0.2	54.3 ± 0.1	54.4 ± 0.1	54.3 ± 0.1	54.3 ± 0.1	54.3 ± 0.1	54.3 ± 0.1	54.3 ± 0.1	54.3
$S = 10$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	44.0	46.2 ± 0.3	46.4 ± 0.2	46.4 ± 0.2	46.4 ± 0.2	46.4 ± 0.1	46.4 ± 0.1	46.5 ± 0.1	46.4 ± 0.1	46.4 ± 0.1	46.4
$S = 20$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	29.1	31.8 ± 0.3	32.1 ± 0.1	32.1 ± 0.2	32.1 ± 0.1	32.2 ± 0.1	32.2 ± 0.1	32.3 ± 0.1	32.3 ± 0.1	32.3 ± 0.1	32.3

(a) LwF [29]

$S = 5$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	50.1	54.7 ± 0.4	54.8 ± 0.3	54.8 ± 0.1	54.8 ± 0.1	54.8 ± 0.1	54.8 ± 0.1	54.8 ± 0.1	54.8 ± 0.1	54.8 ± 0.1	54.8
$S = 10$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	33.7	42.0 ± 0.7	42.1 ± 0.3	42.2 ± 0.4	42.3 ± 0.3	42.2 ± 0.2	42.2 ± 0.2	42.2 ± 0.1	42.2 ± 0.1	42.2 ± 0.1	42.2
$S = 20$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	19.5	27.5 ± 1.4	27.8 ± 0.7	27.8 ± 0.9	28.3 ± 0.4	28.5 ± 0.5	28.6 ± 0.6	28.5 ± 0.4	28.4 ± 0.3	28.4 ± 0.2	28.4

(b) Lucir [21]

$S = 5$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	29.9	31.6 ± 0.2	31.6 ± 0.2	31.6 ± 0.1	31.7 ± 0.2	31.7 ± 0.1	31.7 ± 0.1	31.7 ± 0.1	31.7 ± 0.1	31.7 ± 0.1	31.7
$S = 10$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	22.7	23.8 ± 0.4	23.8 ± 0.2	23.9 ± 0.2	24.0 ± 0.2	23.9 ± 0.1	24.0 ± 0.1	24.1 ± 0.1	24.0 ± 0.1	24.1 ± 0.1	24.1
$S = 20$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	14.8	15.7 ± 0.3	15.7 ± 0.2	15.7 ± 0.2	15.8 ± 0.1	15.8 ± 0.2	15.8 ± 0.1	15.8 ± 0.1	15.8 ± 0.1	15.8 ± 0.1	15.8

(c) Siw [8]

$S = 5$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	28.9	31.9 ± 0.2	32.0 ± 0.1	32.0 ± 0.1	32.0 ± 0.1	32.0 ± 0.1	32.0 ± 0.1	31.9 ± 0.1	32.0 ± 0.1	32.0 ± 0.1	31.9
$S = 10$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	22.6	23.2 ± 0.4	23.5 ± 0.2	23.5 ± 0.2	23.6 ± 0.1	23.5 ± 0.2	23.6 ± 0.1	23.6 ± 0.1	23.6 ± 0.1	23.6 ± 0.1	23.6
$S = 20$	Raw	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$	$R = 7$	$R = 8$	$R = 9$	$R = 10$
	14.5	14.8 ± 0.2	15.0 ± 0.1	15.0 ± 0.2	15.1 ± 0.1	15.0 ± 0.1	15.1 ± 0.1	15.1 ± 0.1	15.0 ± 0.1	15.0 ± 0.1	15.0

(d) FT$^+$ [36]

Table 7.1 – Average top-1 incremental accuracy of *adBiC*-corrected models trained incrementally on Cifar-100 with LwF, Lucir, Siw and FT$^+$, for $S = \{5, 10, 20\}$ states, while varying the number R of reference datasets. For $R \leq 9$, results are averaged across 10 random samplings of the reference datasets. *Raw* is the accuracy of each method without bias correction.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," in *OSDI*, 2015. 31

[2] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *ECCV*, 2018. 16

[3] R. Aljundi, L. Caccia, E. Belilovsky, M. Caccia, M. Lin, L. Charlin, and T. Tuytelaars, "Online continual learning with maximally interfered retrieval," in *NeurIPS*, 2019. 3

[4] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Gradient based sample selection for online continual learning," in *NeurIPS*, 2019. 3

[5] J. Bang, H. Kim, Y. Yoo, J.-W. Ha, and J. Choi, "Rainbow memory: Continual learning with a memory of diverse samples," in *CVPR*, 2021. 3, 17

[6] E. Belouadah and A. Popescu, "IL2M: Class incremental learning with dual memory," in *ICCV*, 2019. 1, 13, 15, 16

[7] ——, "ScaIL: Classifier weights scaling for class incremental learning," in *WACV*, 2020. 15

[8] E. Belouadah, A. Popescu, and I. Kanellos, "Initial classifier weights replay for memoryless class incremental learning," in *BMVC*, 2020. 8, 19, 30, 31, 37, 43, 54

[9] ——, "A comprehensive study of class incremental learning algorithms for visual tasks," *Neural Networks*, 2021. 1, 4, 15, 17, 19, 21, 32

[10] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101 – mining discriminative components with random forests," in *ECCV*, 2014. 33

[11] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," in *ECCV*, 2018. 1, 3, 10, 13, 15, 17, 18

[12] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009. 32, 33

[13] P. Dhar, R. V. Singh, K. Peng, Z. Wu, and R. Chellappa, "Learning without memorizing," in *CVPR*, 2021. 7, 16, 18

[14] A. Douillard, M. Cord, C. Ollion, T. Robert, and E. Valle, "PODNet: Pooled outputs distillation for small-tasks incremental learning," in *ECCV*, 2020. 1, 3, 4, 12, 16, 17, 19

[15] A. Douillard, E. Valle, C. Ollion, T. Robert, and M. Cord, "Insights from the future for continual learning," in *CVPR Continual Learning Workshop*, 2020. 12

[16] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, 1999. 8

[17] B. Fritzke, "A growing neural gas network learns topologies," in *NeurIPS*, 1994. 15

[18] H. He and E. A. Garcia, "Learning from imbalanced data," *Transactions on Knowledge and Data Engineering*, 2009. 1

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016. 8, 10, 31

[20] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NeurIPS Deep Learning Workshop*, 2014. 7, 16, 18

[21] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin, "Learning a unified classifier incrementally via rebalancing," in *CVPR*, 2019. 1, 3, 7, 9, 10, 12, 15, 16, 17, 18, 22, 24, 25, 26, 30, 31, 43, 54

[22] W. Hu, Q. Qin, M. Wang, J. Ma, and B. Liu, "Continual learning by using information of each class holistically," in *AAAI*, 2021. 4

[23] A. Iscen, J. Zhang, S. Lazebnik, and C. Schmid, "Memory-efficient incremental learning through feature adaptation," in *ECCV*, 2020. 10, 17, 19

[24] X. Jin, A. Sadhu, J. Du, and X. Ren, "Gradient based memory editing for task-free continual learning," in *ICML Lifelong Learning Workshop*, 2021. 3

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015. 32

[26] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, 2016. 16

[27] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009. 33

[28] M. D. Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. G. Slabaugh, and T. Tuytelaars, "Continual learning: A comparative study on how to defy forgetting in classification tasks," *TPAMI*, 2019. 15

[29] Z. Li and D. Hoiem, "Learning without forgetting," in *ECCV*, 2016. 1, 3, 7, 10, 16, 18, 22, 24, 25, 26, 30, 43, 54

[30] X. Liu, C. Wu, M. Menta, L. Herranz, B. Raducanu, A. D. Bagdanov, S. Jui, and J. van de Weijer, "Generative feature replay for class-incremental learning," in *CVPR Continual Learning Workshop*, 2020.

[31] Y. Liu, B. Schiele, and Q. Sun, "Adaptive aggregation networks for class-incremental learning," in *CVPR*, 2020.

[32] Y. Liu, Y. Su, A. Liu, B. Schiele, and Q. Sun, "Mnemonics training: Multi-class incremental learning without forgetting," in *CVPR*, 2020. 17

[33] A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," in *CVPR*, 2017. 6

[34] T. Martinetz, S. G. Berkovich, and K. Schulten, "Neural-gas network for vector quantization and its application to time-series prediction," *Transactions on Neural Networks and Learning Systems*, 1993. 15

[35] M. Masana, T. Tuytelaars, and J. van de Weijer, "Ternary Feature Masks: continual learning without any forgetting," in *CVPR Continual Learning Workshop*, 2020. 6

[36] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. van de Weijer, "Class-incremental learning: survey and performance evaluation on image classification," 2021, arXiv:2010.15277. 1, 2, 4, 8, 9, 13, 15, 21, 30, 31, 32, 37, 38, 43, 47, 54

[37] M. Mccloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *The Psychology of Learning and Motivation*, 1989. 1, 8

[38] M. Mermillod, A. Bugaiska, and P. Bonin, "The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects," *Frontiers in Psychology*, 2013. 9

[39] M. Mundt, Y. W. Hong, I. Pliushch, and V. Ramesh, "A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning," 2020, arXiv:2009.01797. 15, 17

[40] S. J. Pan and Q. Yang, "A survey on transfer learning," *Transactions on Knowledge and Data Engineering*, 2010. 6

[41] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019. 15

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019. 32

[43] A. Prabhu, P. H. Torr, and P. K. Dokania, "GDumb: A simple approach that questions our progress in continual learning," in *ECCV*, 2020.

[44] R. Ratcliff, "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions." *Psychological Review*, 1990. 8

[45] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *CVPR*, 2017. 1, 4, 10, 15, 16, 17, 18, 29, 30, 31

[46] N. D. Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni, "Don't forget, there is more than forgetting: new metrics for continual learning," in *NeurIPS Continual Learning Workshop*, 2018. 29

[47] E. Rosch, "Principles of categorization," *Concepts: core readings*, 1999. 33

[48] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-CAM: Why did you say that? visual explanations from deep networks via gradient-based localization," *IJCV*, 2016. 18

[49] A. Soutif-Cormerais, M. Masana, J. van de Weijer, and B. Twardowski, "On the importance of cross-task features for class-incremental learning," in *ICML Theory and Foundation of CL Workshop*, 2021. 8, 10, 12, 15, 46

[50] N. A. Syed, H. Liu, and K. K. Sung, "Handling concept drifts in incremental learning with support vector machines," in *SIGKDD*, 1999. 15

[51] G. M. van de Ven and A. S. Tolias, "Three scenarios for continual learning," in *NeurIPS CL Workshop*, 2019. 3

[52] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *JMLR*, 2008. 10, 11

[53] E. Verwimp, M. D. Lange, and T. Tuytelaars, "Rehearsal revealed: The limits and merits of revisiting samples in continual learning," in *ICCV*, 2021. 17

[54] C. Wu, L. Herranz, X. Liu, Y. Wang, J. van de Weijer, and B. Raducanu, "Memory Replay GANs: learning to generate images from new categories without forgetting," in *NeurIPS*, 2018.

[55] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, "Large scale incremental learning," in *CVPR*, 2019. 1, 2, 13, 15, 16, 18, 21, 23, 24, 30, 38, 47

[56] Y. Xiang, Y. Fu, P. Ji, and H. Huang, "Incremental learning using conditional adversarial networks," in *ICCV*, 2019. 17

[57] L. Yu, B. Twardowski, X. Liu, L. Herranz, K. Wang, Y. Cheng, S. Jui, and J. van de Weijer, "Semantic drift compensation for class-incremental learning," in *CVPR*, 2020. 4, 10, 15, 16, 19, 47

[58] Y. Zhang and Q. Yang, "A survey on multi-task learning," *Transactions on Knowledge and Data Engineering*, 2017. 6

[59] B. Zhao, X. Xiao, G. Gan, B. Zhang, and S. Xia, "Maintaining discrimination and fairness in class incremental learning," in *CVPR*, 2020. 1, 16, 21

[60] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *TPAMI*, 2017. 33