UAB
Universitat Autònoma de Barcelona

# Understanding the Causes of Forgetting in Continually Learned Neural Networks

**Albin Soutif–Cormerais**

Universitat Autònoma
de Barcelona

# Understanding the Causes of Forgetting in Continually Learned Neural Networks

A dissertation submitted by **Albin Soutif–Cormerais** to the Universitat Autònoma de Barcelona in fulfilment of the degree of **Doctor of Philosophy** in the Departament de Ciències de la Computació.

Bellaterra, March 10th, 2024

| | |
|---|---|
| Director | **Dr. Joost van de Weijer**<br>Centre de Visió per Computador<br>Universitat Autònoma de Barcelona |
| Thesis committee | **Gido Van de Ven**<br>Department of Electrical Engineering (ESAT)<br>KU Leuven<br><br>**Bogdan Raducanu**<br>Centre de Visió per Computador<br>Universitat Autònoma de Barcelona<br><br>**Sebastian Cygert**<br>IDEAS NCBR (Poland) |


This document was typeset by the author using LaTeX 2$\varepsilon$.



ISBN: 978-84-126409-9-1

Printed by Ediciones Gráficas Rey, S.L.

# Acknowledgements

First of all, I would like to express my sincere gratitude to Dr. Van de Weijer Joost, my supervisor, for the guidance that he provided me during this thesis. He was very available, supportive and able to bring the best out of me in order to find positive outcomes out of projects that were not always easy to execute. Then, I would like to give special thanks to my numerous collaborators. Antonio Carta who introduced me to the avalanche library, had an impact on my moral that he probably underestimates. Before starting using and contributing to avalanche I was having a hard time both personally and at work, collaborating with the team of people that built this library gave me a fresh look on the field and helped me build real confidence and more discipline in my further projects. Marc Masana was really helpful in the beginning of my thesis and guided me into writing my first article giving very good advices. Bartłomiej Twardowski was, on top of a skilful and very knowledgeable colleague, a very good friend with whom I spent nice times rock climbing. Finally, by the end of my thesis I had the opportunity to collaborate with my friend Simone Magistri that came to Barcelona at a few occasions, as well as his supervisor Andrew Bagdanov. People from the LAMP group, Hector, Alex, Dipam, Kai, Fei, and many others that passed throughout the years, were of great help in understanding and discussing the continual learning literature.

I also thank my family for their unconditional love and support. It was always heartwarming to chat with my parents Philippe and Nathalie and my sisters Lucille and Lison, and a great refilling moment to reunite with them during the holidays. I also want to acknowledge the invaluable help of my friends. A lot of them I met at the workplace, Sergi, Ali, Sanket, Yixiong, Laura, Moha, were here from the start of my thesis and we lived this adventure together. I also want to give a special thanks to the many Italians interns that came to CVC and always brought an amazing atmosphere. Andrea, Marco, Enrico, Pietro, Emmanuele, Francesco, were always keen to go out for tapas after work and made going to work a real pleasure.

Finally, I want to thank all of the amazing people that I met at the different climbing gyms I have been to. I definitely needed this activity to balance the mental-effort with physical one and it was always a pleasure to share a climbing session with friendly people who share a common passion. I give a special thank to Manu and Albert that I met during the early years of this thesis and who have offered great help during the pandemic on top of making me discover the amazing climbing areas of Catalunya.

# Abstract

The use of deep learning has become increasingly popular in the last years in many application fields such as the ones of computer vision and natural language processing. Most of the tasks in these fields are now tackled more efficiently by deep learning than by more classical techniques, provided that enough data is available. However, deep learning algorithms still lack a crucial property, they are not able to efficiently accumulate new knowledge into an existing model. Instead, when learning on new data without revisiting past data they experience *catastrophic forgetting*. This property is the main focus of the sub-field of Continual Learning. The absence of this property leads to various practical consequences. Among them, the computationally expensive nature of learning algorithms that revisit all previously seen data, which comes at a non-negligible energy cost, and privacy issues related to the requirement to store old data for later training.

In this thesis, we investigate the impact of learning in a continual manner on the performance of neural networks, more specifically for classification tasks in computer vision. We investigate the causes of catastrophic forgetting within several commonly studied setups of continual learning. We study the continual learning setting where data associated to distinct set of classes arrive incrementally. Under this setting, we investigate how the difficulty of learning cross-task features accounts for the loss in performance. Part of the thesis is dedicated to the more complex setting of online continual learning, and the problem of the stability gap. We investigate the impact of temporal ensembling on the stability gap and see that we can drastically reduce it by applying an ensembling method at evaluation time, not influencing the training process. In addition, we realise a survey of online continual learning methods and conclude that they might be more affected by an under-fitting problem than by the non-iid training procedure. Finally, we focus on bigger models that have had a strong first learning experience, and study the impact of continual learning on smaller experiences when using low-rank parameter updates.

**Key words:** *deep learning*, *continual learning*, *online learning*

# Resumen

El uso del aprendizaje profundo se ha vuelto muy popular en los últimos años en muchos campos de aplicación como la visión por computador o el procesamiento del lenguaje natural. La mayoría de las tareas en estos campos se resuelven de manera más eficiente usando estas técnicas en comparación con métodos clásicos, siempre y cuando haya suficientes datos disponibles. Sin embargo, los algoritmos de aprendizaje profundo carecen de una propiedad crucial: no son capaces de acumular conocimiento de manera eficiente sobre un modelo existente. En lugar de eso, cuando estos métodos aprenden con nuevos datos sin revisitar los previos, sufren de *olvido catastrófico*. Esta propiedad es el objeto principal de estudio del campo del aprendizaje continuo. La ausencia de esta propiedad conlleva varias consecuencias a la práctica. Entre ellas, el aumento de complejidad de cálculo por la necesidad de incorporar datos prévios para aprender con nuevos datos, lo que conlleva un costo energético importante, o la falta de privacidad por tener que guardar datos antiguos.

En esta tesis, investigamos el impacto del aprendizaje continuo en el rendimiento de las redes neuronales, más específicamente en tareas de clasificación usando visión por computador. Investigamos las causas del olvido catastrófico dentro de varios escenarios comúnmente estudiados en el aprendizaje continuo. Analizamos el entorno de aprendizaje continuo donde los datos asociados a conjuntos de distintas clases llegan incrementalmente. En este contexto, investigamos cómo la dificultad para aprender características entre tareas afecta la pérdida de rendimiento. Parte de la tesis se dedica al entorno más complejo del aprendizaje continuo en línea y al problema de la brecha de estabilidad. Estudiamos el impacto de los conjuntos temporales de modelos en la brecha de estabilidad y observamos que, durante la evaluación, podemos reducirla drásticamente aplicando un método de ensamblado de modelos, sin influir en el proceso de entrenamiento. Además, realizamos una revisión de métodos de aprendizaje continuo en línea y concluimos que pueden verse más afectados por un problema de subajuste que por el procedimiento de entrenamiento con un flujo de datos no independiente e idénticamente distribuido (i.i.d). Finalmente, nos centramos en modelos más grandes que han tenido una experiencia de aprendizaje inicial sólida, y estudiamos el impacto del aprendizaje continuo en experiencias más pequeñas al usar actualizaciones de parámetros de bajo rango.

**Palabras clave:** *aprendizaje profundo, aprendizaje continuo, aprendizaje en línea*

# Resum

L'ús de l'aprenentatge profund ha crescut en popularitat els darrers anys en molts camps d'aplicació com el de la visió per computador i el processament del llenguatge natural. Actualment, la majoria de les tasques en aquests camps es resolen de manera més eficaç mitjançant l'aprenentatge profund que amb tècniques clàssiques, sempre que hi hagi prou dades disponibles. No obstant això, els algorismes d'aprenentatge profund manquen encara d'una propietat crucial, no són capaços d'acumular eficaçment nous coneixements en un model existent. En lloc d'això, quan aprenen amb noves dades sense revisar les dades passades, experimenten un *oblit catastròfic*. Aquesta propietat és el principal focus dels subcamps de l'Aprenentatge Continu. L'absència d'aquesta propietat porta a diverses conseqüències pràctiques. Entre elles, la naturalesa computacionalment cara dels algorismes d'aprenentatge que revisiten totes les dades vistes anteriorment, el que comporta un cost energètic no negligible, i els problemes de privadesa relacionats amb la necessitat d'emmagatzemar dades antigues per a un entrenament posterior.

En aquesta tesi, investiguem l'impacte de l'aprenentatge continu en el rendiment de les xarxes neuronals, en especial, per a tasques de classificació en visió per ordinador. Investiguem les causes de l'oblit catastròfic dins de diversos escenaris comuns d'aprenentatge continu. Estudiem l'entorn d'aprenentatge continu on les dades associades a conjunts diferents de classes arriben incrementalment. En aquest context, investiguem com la dificultat d'aprendre característiques entre tasques explica la pèrdua de rendiment. Part de la tesi està dedicada a l'entorn més complex de l'aprenentatge continu en línia, i al problema de la pèrdua d'estabilitat. Investiguem l'impacte de l'agrupament temporal en la pèrdua d'estabilitat i observem que podem reduir-lo dràsticament aplicant un mètode d'assemblatge durant l'avaluació, sense influir en el procés d'entrenament. A més, realitzem una revisió dels mètodes d'aprenentatge continu en línia i concloem que podrien estar més afectats per un problema de subajust que pel procediment d'entrenament no iid. Finalment, ens centrem en models més grans que han tingut una primera experiència d'aprenentatge sòlida, i estudiem l'impacte de l'aprenentatge continu en experiències més petites utilitzant actualitzacions de paràmetres de baix rang.

**Paraules clau:** *aprenentatge profund, aprenentatge continu, aprenentatge en línia*

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Deep learning of artificial neural networks

In the recent years, Deep Learning techniques that automatically learn the weights of an artificial neural network from data, using the back-propagation algorithm [138], have unlocked drastic improvements in a broad range of fields [89]. Computer vision [136], Natural language processing [20], Speech recognition [64], Robotics [11], Environmental Modelling [87], Bio-informatics [78], and many others have seen exciting developments thanks to advances in this field. The training process of most of these advanced deep learning systems follows a simple scheme. Once the training data is collected and cleaned, it is shuffled and presented to the model in small chunks that are called *mini-batches*. The model then makes a prediction, computes the difference between its prediction and the actual label (in case of supervised learning), and then slightly tunes its weights in order to improve the prediction. Going through the whole training dataset several time, until the performance on a held-out validation dataset converges, results in proper weights that can be used to solve the desired task. Although this process seems easy to implement, it leads to rigidity in the ability of the model to acquire new information. Indeed, if a new portion of training data arrives, naively training on it following the same process will most probably not lead to better weights and might even decrease the performance on the previously seen data. This phenomenon is known as *catastrophic forgetting* [53, 59]. This inability to accumulate new knowledge has serious consequences in terms of computational and memory requirements of learning, since updating a model with new data then requires to store all the previous training data and to revisit it jointly with the new data.

As a matter of fact, in terms of energy consumption, it is estimated that a big language model like ChatGPT-3 can consume from 1 to 10 gigawatt-hour* (GWh) of energy for training, which is equivalent to the yearly electricity consumption of over 1,000 U.S households [116, 126]. In these extreme cases, it is evident what would be the positive impact of a strategy that mitigates the cost of assimilating new data into the model, since training from scratch on the accumulation of previous and new data would be an energetic nightmare, that needs to be repeated whenever a model update is required.

*This estimation differs a lot when taken from different sources

In terms of memory, while it is true that the complete storage of even recent large scale datasets is cheap comparatively to the computation required to learn on it [130], trying to revisit every previously seen training instance while learning on new data comes with additional problems. Firstly, as the memory size grows it becomes more and more expensive to revisit it while learning a new task, so revisiting a bigger memory also impacts the computation leading to the same consequences that are mentioned in the previous paragraph. Secondly, a lot of applications also require to give privacy guaranties [99], in which case retaining specific training instances would not be possible for those instances that are considered private.

The research field of *Continual Learning* [93, 96] is dedicated to finding solutions to the above-mentioned issues, it studies and aims to improve the efficiency of learning in an incremental setting under the constraint of restricted memory and computation. It will be the core focus of this thesis.

## 1.2 Continual learning

### 1.2.1 Definition

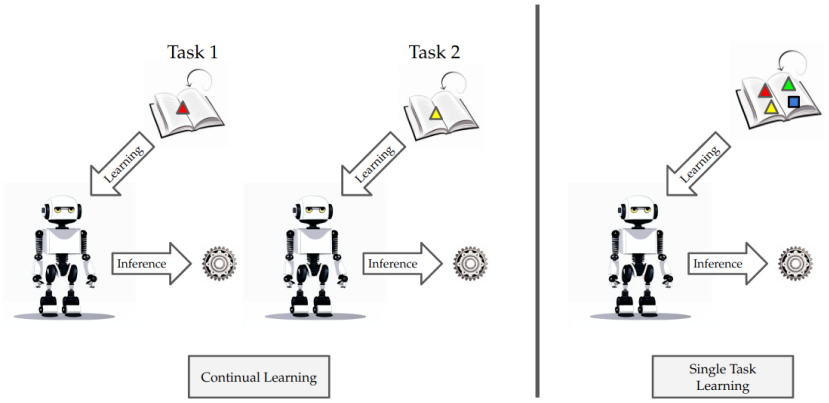


Figure 1.1: Schema highlighting the difference between Continual Learning and Single Session Learning. The Continual Learning agent is deployed for inference before finishing learning and progressively assimilates new concepts.

Continual learning describes the process of learning from a stream of data by progressively incorporating new knowledge. It is opposed to learning from scratch

in a single session on an independently and identically distributed (i.i.d) dataset, we highlight this difference in Figure 1.1. Every time new information arrives, it must be built on top of previously learned knowledge, without forgetting the previously acquired one. This approach to learning aims to be closer to the human learning process, since we are also exposed to experiences in an incremental manner. In practical terms, the ability to continually learn should also lead to more efficient learning in terms of computation and memory.

Formally, continual learning can be thought of as learning from a data stream with potentially varying data distribution. In this thesis, we will mostly talk about supervised learning problems, where we need to predict a label $y_t \in \mathscr{Y}$ from an input $x_t \in \mathscr{X}$ at time $t$. In that case, continual learning considers the setting where $x_t$ and $y_t$ are drawn from a distribution that can vary with time $x_t, y_t \sim p_t(x, y)$. After learning, or in-between the learning sessions, the learner is expected to predict labels from inputs coming from a test distribution that is often taken as the joint distribution over all timesteps $x_{test}, y_{test}, t_{test} \sim \frac{1}{T}\sum_0^T p_t(x, y)$, where $t_{test}$ is the timestep that identifies the distribution mode from which $(x, y)$ has been drawn, $t_{test}$ is later referred to as the *task label* or *task id*. Let's define a prediction function $f : \mathscr{X} \times \Theta \longrightarrow \mathbb{S}^C$, with learnable parameters $\theta \in \Theta$. Here $C = Card(\mathscr{Y})$ and $\mathbb{S}^C$ is the probability simplex on $\mathbb{R}^C$ s.t $\mathbb{S}^C = \{x \in \mathbb{R}^C : \sum_{i=0}^C x_i = 1, x_i \geq 0 \forall i\}$. We further denote $\theta_t$ as the parameters learned up to time $t$, and define one timestep as the arrival of a set of new inputs that will be learned in a single session. Then, a continual learning algorithm $\mathscr{A}$ learns the parameters $\theta_t$ given the previous parameters $\theta_{t-1}$ and the dataset $\mathscr{D}^t$ for time step $t$ as described in Equation 1.1.

$$\theta_t = \mathscr{A}\left(f, \theta_{t-1}, \mathscr{D}^t\right) \tag{1.1}$$

This problem is inherently difficult because the quantity to optimize is most often not directly accessible to the algorithm, since it only sees part of the full training distribution at a time. As a consequence, a continual learning algorithm often attempts to prevent *catastrophic forgetting* of previous knowledge, by using information coming from the previous weights, or from some externally stored memory that allows to estimate the loss function on previously seen tasks. If we denote $\mathscr{L}^t : \mathbb{S}^{\mathbb{C}} \times \mathscr{Y} \longrightarrow \mathbb{R}$ the loss function for task $t$, then the ultimate goal of the algorithm is to optimize the following problem

$$\underset{\theta}{\operatorname{argmin}} \frac{1}{T} \sum_{t=1}^{T} \mathbf{E}_{x,y \sim p_t(x,y)}[\mathscr{L}^t(f(x, \theta), y)] \tag{1.2}$$

### 1.2.2 Settings and Benchmarks

Many benchmarks have been proposed in the literature, each of them defining the sequence of training distributions $p_t(x, y)$ and the prediction task in a different way [152]. In this section we will present the different kinds of benchmarks and their corresponding challenges.

**Class- and Task- Incremental:** In supervised learning, a common practice to create continual learning benchmarks is to split the training data of an existing supervised benchmark (i.e MNIST [90], Cifar100 [85], ImageNet [139]) based on mutually exclusive sets of classes (one class that is in one set cannot be in another set). This practice leads to the two most common benchmark type, the *class-incremental learning* (CIL) and the *task-incremental learning* (TIL) benchmarks. In *task-incremental learning*, both $x_{test}$ and $t_{test}$ are available at test time, which gives the option to split the learning into different components that will or will not be activated for a given task. However, in *class-incremental learning*, only $x_{text}$ is available at test time, rendering these kinds of strategies impossible or significantly harder (it is still possible to first estimate $t_{test}$ then $y_{test}$). More formally, in *task-incremental learning*, the prediction function $f$ is trying to approximate $p_t(y|x) = p(y|x, t)$ whereas in *class-incremental learning*, it is trying to approximate $p(y, t|x)$. These two settings are the most commonly explored because they can be easily created from existing labeled datasets. Extensive evaluation of existing methods for Class-incremental learning can be found in [111], and for Task-incremental learning in [40].

**Domain Incremental:** Domain incremental benchmarks are not created based on the label of the data, but rather based on its input domain. For instance, it can be images with different lightning conditions either natural or artificial, or with different sets of artificial augmentations applied to images of each task (Rotated MNIST or Permuted MNIST [59]). This kind of benchmark is slightly less popular due to the easiness of performing additional transformations during training to counter the input domain shift. For instance, in the case of rotated MNIST, applying a random rotation to incoming images would make the learning similar to the one on an i.i.d stream. However, learning on natural domain shifts is an interesting problem which is actively explored. Instances of natural domain shift can be due to the time and place of where a picture was taken [24], or even more drastic shifts can be obtained with different kind of modalities representing one category (real image and different kinds of visual representation of this same category) [127].

**Offline and Online Learning:** Among the different kind of benchmarks, another distinction is often made in the field of continual learning between *online learning*

and *offline learning*. This terminology distinguishes two kind of benchmarks based on the amount of data that is available while training a new "task". For the purpose of explaining this concept more in detail, we will use the term of *experience* as done by Lomonaco et al. [103], to refer to an incoming dataset, while we will keep the term *task* to indicate a distribution shift and point to a specific data distribution. Using this terminology, we can say that offline learning has access to experiences of significantly bigger size than *online learning*, which are in general big enough to split them in smaller chunks (mini-batches) and perform several epochs over them, while *online learning* can only access one small chunk of data at a time, while it is still possible to perform multiple iterations over it, it is in general less interesting performance wise. While it is often believed that *online learning* is computationally cheaper than *offline learning*, this is not a condition required by the online learning setting. However, the inefficient aspect of performing a big number of iterations on a small batch of data often leads to cheaper algorithms. Typically, *online continual learning* is of interest in settings where the data arrives at a very fast pace and cannot be processed fast enough by an offline learning system. This can happen in sensor networks [10], with data coming from the Internet of Things, or in financial applications [123].

### 1.2.3 Desired properties

In this section, we will quickly review the main properties that we expect a continual learning agent to have. Here by continual learning agent, we refer to the ensemble of model, weights and continual training algorithm described in the previous section. The properties listed below are largely inspired by observations made on human learning, and have been presented under different formulations in various existing works [30,93]. For now, and because this is still an open problem, we will not directly link these properties to metrics, but rather describe them in qualitative terms.

**Knowledge Transfer:** Whenever a continual learning agent encounters a new task, it should be able to exploit the knowledge that it has accumulated so far to quickly adapt to this new task and perform better on this task compared to an agent that has no experience, this capacity is commonly referred to as *forward transfer*. We also expect this knowledge transfer to work backwards, when an agent trains on a new task it should ideally improve on the previous tasks, or at least do not forget them, this latter property is referred to as *backward transfer*. If learning a task hurts the capacity of an agent to solve previous tasks, then it suffers from *forgetting* [53,59].

**Memory and Computational Efficiency:** The goal of continual learning is to accumulate the knowledge from a stream of task without forgetting the previously

learned knowledge. This definition does not necessarily imply that the continual learning agent should not revisit previous data. And revisited limited amounts of data is a method that is often used in continual learning. However, a caveat of this definition is that the agent would then not be forced to accumulate knowledge from each task but could instead perform single-session learning by storing all of the data seen so far and learning from scratch again every time a new datapoint or set of datapoints arrives. This would in turn not make it a continual agent anymore.

In order to impose methods to perform actual continual learning with progressive accumulation of knowledge, the research community came out with artificial constraints that bound the memory and/or the computation of a method. This can be done for instance by limiting the amount of previous data an agent can store to a fixed amount, or it's amount of computation to a fixed amount. In practice, it is easier to impose a constraint on the memory because it is a component that is easier to control (especially under fixed network architecture and with a memory that depends mostly on the amount of input data stored). The memory bounding is also highly inspired by what we understand from the human learning. While it is not clear what amount of "computation" we use, it seems quite clear to us that we cannot recreate to the perfection every experience we have lived, which means that we do not store the equivalent of input datapoints, or at least we discard a significant proportion of them. This is why most often the memory bound constraint is used, often disregarding the computational constraint.

However, this has been recently put in question [130] since from a practical point of view, memory is very cheap in regard to computation. In terms of computation, we expect a continual agent to learn a new task faster than it would take to learn from scratch this task plus all previous tasks. If we denote the computation of learning task $t$ (alone) as $C_t$, then we want the learning agent to assimilate task $k$ faster than $\sum_{t=1}^{k} C_t$, which corresponds to learning from scratch on all tasks seen so far. In general, continual learning methods reach computation budgets that are of the same order than the one of learning from scratch on all the tasks only once, hence they aim to assimilate task $t$ using approximately $C_t$ computation. This means that they aim to have little to no additional computation inherent to the use of a continual learning method compared to the case where all of the data would be available from scratch. Of course, this is difficult to achieve.

**Stability vs Plasticity:** Although retaining the previously learned knowledge is important, it is also important that the network is capable to correctly learn the new incoming tasks. The term of *plasticity* is used to refer to the capacity of adaptation of a model, and is often contrasted with the term of *stability*, that refers to the ability to solve previously learned tasks at any time during the learning. This interplay

between the two properties of *plasticity* and *stability* is referred to as the *stability-plasticity tradeoff*, since for many methods, gaining in plasticity results in losing stability. Attempts at designing plasticity metrics [105] have been made, but it is in general quite hard to measure this property, so people often use *intransigence* [28] as a proxy measurement, which measures the difference in performance between the continually learned model and the "ideal" model learned on the full stream of data. Note that this latter metric cannot be computed in a realistic setting but can only be used to evaluate a continual learning method in a controlled setting where access to the full dataset is possible.

### 1.2.4 Continual learning in the wider Deep Learning landscape

In order to situate continual learning in the wider field of deep learning, we here take a look at the related fields from which continual learning research draws its inspiration. While each chapter will contain a specific related works section more specific to continual learning, in this section we will talk about fields that do not directly tackle continual learning but are of interest for its study and are often taken as inspiration to design continual learning methods. In Figure 1.2, we visualise an undirected citation graph of the bibliographic resources used during this thesis, on which we run a community detection algorithm to identify sub-fields of papers that are highly connected between each other and weakly connected to the rest of the papers. While articles are clustered in a big connected component because very related to continual learning (on the right), we can distinguish several other clusters that are only weakly connected to this big component and are not directly continual learning papers.

**Compositional Learning:** This field explores the learning of compositional or modular structures, with the hope that individual components will be able to learn meaningful functionalities that can be used and composed on demand when a task requires it. If such meaningful functionalities are found, modular learning promises for fast adaptation to unseen tasks and improved inference speed and memory. While it is believed that this modularity can emerge from normal neural network training [37], this property can be quite hard to harness as-is. For this reason, most works in this field attempt to force modularity by learning a composition of smaller neural networks along with gates that manages the routing of the module's output into another module input [122, 145]. While initially explored using a single level of modularity, with one gate that routes the input to a dedicated expert network [145], leading to a "Mixture of experts", more recent works explored multiple levels of modularity [114]. Advances in this field are of high interest for continual learning because of the improved capacity of compositional networks to isolate the knowledge required to solve a task from the rest of the knowledge. This isolation of knowledge should lead to increased

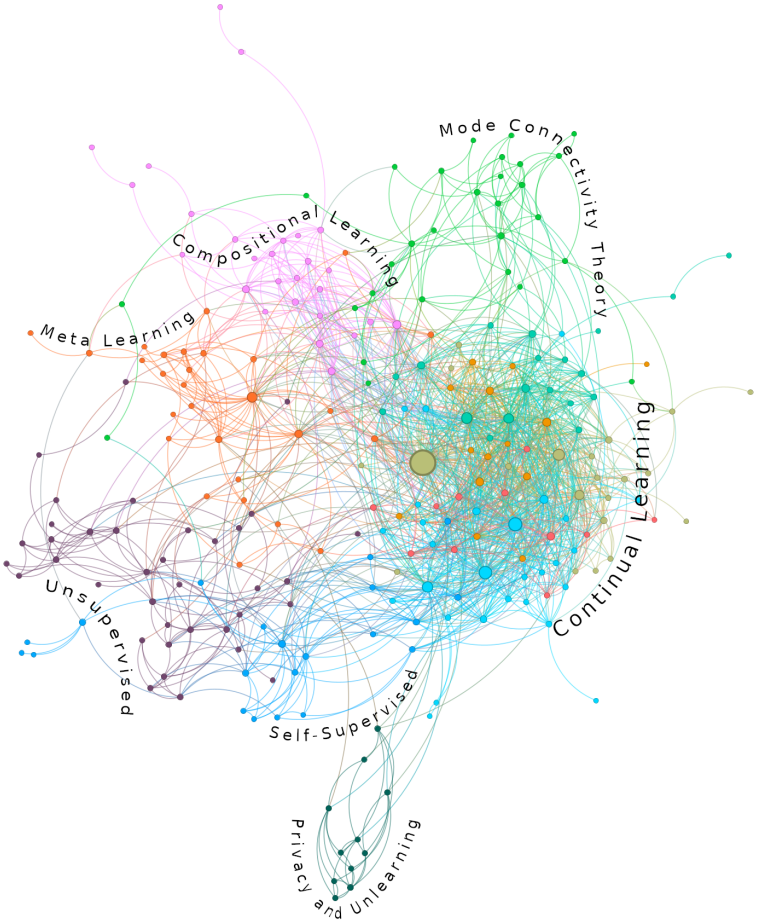


Figure 1.2: Undirected citation graph of the bibliographic resources explored during this thesis. The main cluster on the right corresponds to purely continual learning papers, while the other clusters correspond to other related sub-fields of deep learning.

protection of remaining weights, and in the ideal case, routing the correct part of the input to the correct module could lead to module-wise i.i.d training, which is a dream for any continual learning practitioner. For these reasons, multiple continual learning works explored this avenue starting with the simple mixture of experts [7], to the more complex multi-level routing [113, 122]. Although they work well on task-incremental learning benchmarks, a known issue of these methods is that they struggle in class-incremental learning especially when the task-inference is difficult to perform, which often makes the routing of inputs to the correct modules harder. Interestingly, however, mixture of experts have proven to be extremely efficient in recent large language models (LLM) [77] even when the "dedicated functionality" of each component remains hard to interpret.

**Meta Learning:** This is a sub-field of artificial intelligence that investigates the possibility to improve the learning capabilities of a given algorithm. Most approaches do that by differentiating through the backpropagation process, which requires additional computation, although some alternatives have been found to save computation [118]. Using this technique, they can learn any parameter that are part of the classical training process (i.e. hyperparameters), such as the initial weights [51], the loss [15], or any other parameter introduced by the specific training algorithm used. Since the classical learning process leads to *catastrophic forgetting* in the setting of continual learning, many attempts have been made at using Meta-Learning in order to improve the continual learning capabilities of an agent [29, 63, 137], with the hope that the meta-learning process can learn a more effective way of performing continual learning.

**Mode Connectivity Theory and the impact of Early training phase:** Mode connectivity refers to the possibility of finding a path in the parameter space between two different optimums, such that every point of this path is of low loss. Initially studied in Garipov et al. [55], it has been quickly linked to interesting properties of the early training phase of neural networks. In particular, it has been shown that very early in training, a neural network reaches a point that is stable to SGD noise, meaning that starting the training from that point on a given dataset but with different optimization trajectories will always lead to linearly connected minima [52]. Parallel studies have shown that perturbing the training of the network early in training could lead to permanent loss of performance on subsequent tasks [2]. Studying the impact of the early training phase of neural networks on the subsequent learning steps is crucial for continual learning, since a continual learning agent is expected to learn on a sequence of tasks by always starting from the previous model checkpoint. Although some conclusions from this field might seem catastrophic news for continual learning, in particular the results showing that incremental learning on incrementally bigger

i.i.d datasets lead to reduced performance [12], it is not clear yet how these results can generalize to different optimization configurations (higher learning rate, presence of weight decay, different optimizer), and the impact of the early phase of training in continual learning is still poorly understood. The mode connectivity property has already been studied in the continual learning setting [115]. In general, most continual learning works assume that the difficulties of continual learning only comes from learning on non-i.i.d sequence of datasets, but works on the early phase of training put that in question, showing that incrementally training on top of a previous model checkpoint might lead to unexpected difficulties in itself.

**Unsupervised and Self-Supervised Learning:** These are learning paradigms that try to alleviate the need for labeled data, by learning representations solely from unlabeled inputs. Self-Supervised learning refers to the more specific class of unsupervised algorithms that in order to learn representations, create a pretext task, for instance via transforming the data in some way and trying to predict what transformation has been applied [56], or by performing multiple transformations of the same input and making their output feature match [32]. Variations of these methods also allow for learning representations using a mix of labeled and unlabeled inputs [80]. Continual learning has been mostly explored in the supervised setting, however because of the incremental nature of the supervised task, it is not obvious whether the use of a purely supervised loss in that setting is optimal. Indeed, in the class-incremental learning setting, learning each task individually with a supervised loss struggles to learn cross-task features, as we will detail in Chapter 2. For this reason, attempts have been made at using losses from the field of self-supervised learning in continual learning, either in the supervised setting [109] or the unsupervised one [57]. But more importantly, unsupervised learning introduced great tools for evaluating the strength of the representations, that are now used and have given interesting insight on the representations learned in continual learning [4, 38].

### 1.2.5 Open source implementations

Reproducibility of existing experiments is a common concern in many fields. Fortunately, computer science is one of the fields where reproducibility has a central position, however, it does not mean that it should be taken for granted. While it is very easy to reproduce a deep learning experiment by running the same code that has been run by the author, it is not easy to ensure that the code written by an author reflects what has been explained in the corresponding article. In particular, in deep learning, it is common to obtain improved results by tuning the hyperparameters of a method more than the ones of another method. Or by modifying some parts of the training code for one method but not the equivalent part for the baseline method. All of these

considerations make it hard to conclude on the ability of a method to improve the results on a given benchmark or not, especially when this improvement is marginal (which occurs for many articles). This is why it is important to spend time gathering all the proposed methods or benchmarks under a common framework in order to make their comparison easier and more fair. On top of that, it often makes comparison and prototyping of new methods easier. A variety of open-source libraries have been proposed and are concurrently used in the literature. Most of these libraries emerged from the effort of individual students after having implemented multiple methods for the purpose of making a survey paper, like FACIL [111] and PyCIL [170, 171], these libraries implement most aspects of the continual learning training and evaluation pipeline as well as a considerable amount of methods. They put the focus on exactly reproducing methods as they have been designed in their initial respective papers (at the cost of less code sharing between methods). Continuum [49] is a library that focuses essentially on making the creation of continual learning benchmarks easy, by providing utilities for splitting datasets into multiple tasks and manipulating the task specific datasets.

**The Avalanche library** [26] is an end-to-end continual learning library that implements all aspect of a continual learning pipeline (See Figure. 1.3) as well as its evaluation. It is available on github at https://github.com/ContinualAI/avalanche.git and has more than 1600 stars on github and 60 collaborators, which makes it the most popular continual learning library at the time of writing this thesis. This library puts the focus on making prototyping of new methods easier and more natural, which goes together with sharing a maximum amount of code between methods. A part of this thesis has been dedicated to developing new functionalities for this library.

## 1.3 Objectives and Approach

In this thesis, we study the continual learning of neural networks and try to uncover the causes of the performance drop in this setting, which are classically regrouped under the naming of *catastrophic forgetting* but in reality result from a more complex set of causes that we attempt to pinpoint in the different chapters. In this section, we describe the methodology that leads to the writing of each chapter, by specifying the problem that was encountered and the resulting objective that we define.

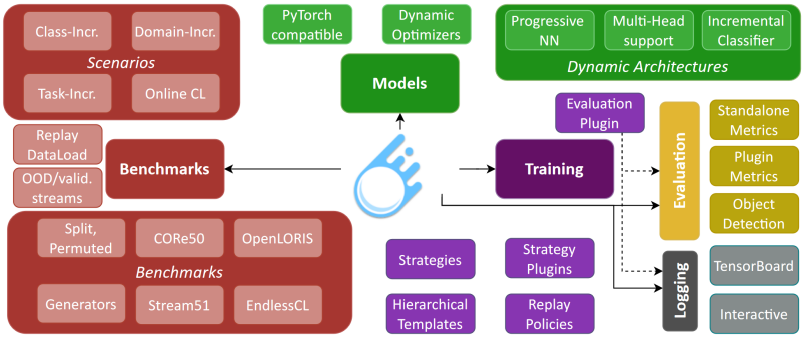


Figure 1.3: Architecture of the avalanche library (taken from [26]), giving an idea of all the components that enter into the designing of a continual learning training and evaluation pipeline.

### 1.3.1 On the importance of cross-task features for class-incremental learning

In the *class incremental learning* setting, *catastrophic forgetting* is in general even more pronounced than in *task incremental learning*. This is due to multiple reasons some of them are studied in the literature. However, forgetting is still largely mentioned as a cause of performance drop in that setting, even when a reasonable amount of data is retained to be later replayed which usually leads to a huge reduction of forgetting in task incremental learning [31]. To shed light on the differences between task and class incremental learning, we decide to ablate the learning of cross-task features in class-incremental learning, and we look at the performance that can be obtained without this component. This allows us to highlight one big difficulty that methods need to overcome in order to solve the task of class-incremental learning. This can be summarized through the following objective:

> **Ablation of the learning of cross-task features in class-incremental learning:** Study a yet unexplored cause of performance gap in class-incremental learning and compare its influence to the one of other causes that happen in task incremental learning.

### 1.3.2 Improving Online Continual Learning Performance and Stability with Temporal Ensembles

The phenomenon of *stability gap* is a new interesting observation that has been made in the literature, initially in the setting of *online continual learning* [22, 88]. It is based on the observation that upon starting the training of a new task, the performance of the model initially decreases on previous tasks (somewhat drastically) before going back up to a higher value. While it is not yet known if this phenomena durably impairs training, several metrics have been proposed to measure it quantitatively [88]. In order to mitigate it, we propose to investigate the usage of model ensembling methods used at evaluation. We compare different ways of implicit ensembling that are reasonably applicable with a minor memory overhead and study the impact of such methods on the stability gap as well as other continual learning metrics. This is summarized through the following objective:

> **Study the impact of temporal ensembling at test time in online continual learning:** experiment with different implicit temporal ensembling methods and evaluate their impact on several metrics in online continual learning. We hypothesise and verify that temporal ensembling should mitigate the stability gap.

### 1.3.3 A Comprehensive Empirical Evaluation on Online Continual Learning

Online class incremental learning is a setting that imposes stronger constraints on the availability of the data for training, by considering each incoming mini-batch as a task. Training in this setting generally leads to lower performance. Although this is a well studied field, it is still not clear by what mechanism it is possible to increase the performance in this setting, and whether the lack of performance of continual learning methods comes from the non i.i.d nature of the stream or from the lack of computation. In order to understand more about these aspects, we perform an empirical evaluation of several existing methods and confront them to a wide range of continual learning metrics. In particular, we choose to report metrics that reveal the representational power by ignoring the additional difficulty of maintaining an updated classification head that is encountered in non i.i.d settings. This set of metrics helps us conclude on the current problems of online continual learning methods. We set ourselves the following objective:

**Perform an empirical evaluation of existing online continual learning methods:** Use a wide range of metrics to better describe existing method and find out what advantages and limits current methods have. Compare the performance of the continual learning methods to the one of naive training using the same amount of computation on the i.i.d stream.

### 1.3.4 An Empirical Analysis of Forgetting in Pre-trained Models with Incremental Low-Rank Updates

The emergence of the use of large pretrained models in practical deep learning applications has lead to the wider use of low-rank training techniques, that aim to optimize in a lower dimensional parameter space that is later projected back into the full parameter space. So far, the interplay between these techniques and continual learning has not been explored, we hence analyze how the application of these techniques affect various continual learning metrics. This is summarized through the following objective:

**Evaluate the impact of low-rank updates in continual learning:** Progressively increase the rank of a widely used low-rank learning method and study its impact on continual learning metrics. Study its impact under different architectures and make new observations on forgetting in large pretrained models.

# 2 On the importance of cross-task features for class-incremental learning*

## 2.1 Introduction

A particular challenge of class-incremental learning (class-IL) is that classes in different tasks are never learned together but have to be discriminated from each other (no task-ID at test time). One way to tackle this is to store a small subset of instances of each class seen so far to later do rehearsal when learning new ones [27, 135]. Therefore, the agent is capable of seeing classes from different tasks together over the course of training. Even though some class-IL methods do not make use of a memory buffer [43, 165], these can currently not compete with methods using one [111]. Therefore, in this chapter, we focus our attention on methods with a memory buffer. However, because of the memory restrictions of such rehearsal memory buffer, a class imbalance is introduced during the learning process. One of the side effects of this class imbalance is that the biases and the norm of weights in the classifier tend to be higher in the classes from newer tasks [69, 111, 163]. This phenomenon is called task-recency bias and is tackled with different bias-correction strategies by recent approaches [16, 69, 163]. Having addressed the task-recency bias, there is still a large gap with respect to the upper-bound (joint training). In this chapter, we investigate to what extend the training of better cross-task features can narrow this gap.

Another focus of this work is the study of the causes of performance drop in class-IL. As in other incremental learning settings, processing the sequence of tasks in class-IL leads to a dramatic drop in performance from earlier tasks to the latest ones. Many class-IL works associate *catastrophic forgetting*, which has been widely studied in the task-incremental learning (task-IL) setting, to this performance drop. However, the metrics defined for task-IL are not well suited to the class-IL setting and can give misleading results when applied directly to the latter. In this work, we will define similar metrics adapted to class-IL, and use them to increase understanding of the main causes of performance drop when moving to that setting.

When learning in an incremental manner, we consider two types of features that can be learned by the feature extractor (see Fig. 2.1). A *cross-task discriminative feature* discriminates between classes that belong to different tasks, while an *intra-task*

*This chapter is based on an article in the International Conference on Machine Learning (ICML) Workshop on Theory and Foundations of Continual Learning, 2021 [149]

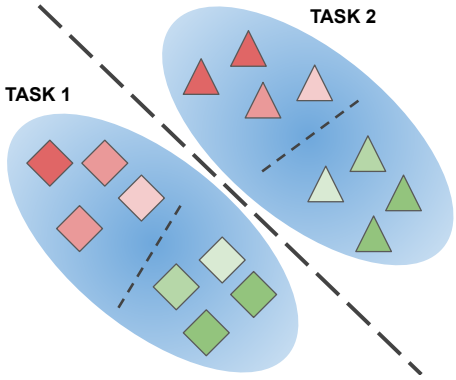


Figure 2.1: Fictive scenario illustrating the two types of features considered. In this scenario, the color is an intra-task feature, and the shape is a cross-task feature. The intra-task features are insufficient to solve the final 4-class problem.

*discriminative feature* discriminates between classes from the same task. Furthermore, some features might appear during training that satisfy both types of discrimination. Given feature types, we postulate that replay in class-IL should fulfil multiple roles at the feature extractor level. First, to maintain previously learned intra-task discriminative features. Second, to create cross-task discriminative features capable of discriminating between classes not present in the same task. For instance, in Fig. 2.1, discriminating between classes of the same task only requires to learn color features (intra-task), while solving the cumulative tasks proposed in class-IL requires to also learn shape features (cross-task). Its third role is to enable knowledge transfer between the new task and previous tasks so that one task can benefit from the learning of another as it occurs when learning multiple tasks concurrently [14]. Failing to learn either type of feature would result in higher miss-classification rate. In this chapter, we aim to study whether these two types of features are learned properly with replay, especially cross-task features.

Our contributions are summarised as follow:

- We compare two baselines using replay. One aims to learn cross-task features while the other does not. This helps us to assess what is the importance of cross-task features in the class-IL setting.

- We question the use of classic task-IL metrics in class-IL and propose new appropriate metrics. We observe that *catastrophic forgetting* is not the main cause of performance drop in class-IL.

In addition, we perform experiments allowing an increasing amount of memory to the class-incremental algorithms. These results suggest that even though there is no forgetting, there is room for improvements in task specific accuracy that can significantly improve cross-task discrimination without explicitly building features for it.

## 2.2 Related Work

Class-incremental learning approaches can be divided in three categories [111]: regularisation-based, bias-correction, and rehearsal-based, which are often combined to tackle multiple incremental learning challenges.

Regularisation-based approaches add a regularisation term to the loss which penalises changes in the weights [5,28,82] or activations [79,96,134,167]. This maintains the stability-plasticity trade-off between maintaining previous knowledge and learning new tasks. These approaches have been widely used in task-IL. Furthermore, they have also shown promising results in class-incremental learning, whether by pairing with rehearsal-based approaches [27, 69, 135, 163] or with other strategies such as attention [43].

Rehearsal has become the most commonly used approach in class-incremental learning, focusing on images or feature replay of previous tasks while learning a new one. Rehearsal-based approaches can be further divided into different subcategories. Pseudo-rehearsal methods replay generated images [146], thus avoiding the privacy issue of storing raw data. They also have the advantage of reducing storage memory and generating it on-the-fly instead. In MerGan [161], a conditional GAN is used to replay images, introducing the learning and storage of an advanced network instead of having to store exemplars. In Generative Feature Replay [100], a GAN is used to replay smaller size features in to the classification layer, making the GAN learning easier at the cost of fixing the feature extractor, limiting the power of future learning. Some pseudo-rehearsal methods generate images by directly optimising in the image space using a loss that aims to prevent forgetting [76, 101], thus avoiding the use of an external generator. Classical rehearsal methods store exemplars and replay them along with the data from the task at hand. However, this introduces a data imbalance problem between the few exemplars available for previously learned classes and the large amount of data for the new ones. For that reason some approaches combine rehearsal with bias correction [16, 69, 163], which also tackles task-recency bias. In IL2M [16], a dual-memory is proposed, storing both images and class-statistics, which are used to rectify the scores of past classes. Noticeably, most rehearsal approaches use a distillation loss term on the outputs of the model, while IL2M obtains similar results with a classic fine-tuning cross-entropy loss instead.

A more challenging setting is introduced with online learning, where no more than one pass is allowed on the current task data. Therefore, instead of incremental training sessions which can iterate over several epochs on the data, each sample is seen once unless stored in the reduced external exemplar memory. This setting was originally explored for task-IL, where the task-ID is known at test time [30, 104]. In GEM [104] and A-GEM [30], gradients are modified in order to avoid both increasing the loss on the exemplars and over-fitting on them. Eventually, it is observed in *tiny episodic memories* [31], that just replaying exemplars, even when only a few are available, efficiently prevents forgetting in the task-IL setting. Recently, some approaches have been applied to online class-IL [6,9,67]. In MIR [6], controlled sampling is performed to automatically rehearse samples from tasks currently undergoing the most forgetting. REMIND [67] compresses features from the exemplars using learned quantization modules, which later are replayed.

In recent surveys [17, 111], inter-task confusion and bias correction are identified among the main challenges of class-IL. We argue that additional challenges are the ones of knowledge transfer between tasks, and the learning of cross-task features (in contrast with task-recency bias). Finally, GDumb [131] introduces a baseline which only uses exemplars from the buffer, obtaining comparable performance to the state-of-the-art in the online setting. The proposed method also works for class-IL since it only needs the exemplars present in the memory buffer, however, it achieves a lower performance. Similarly, our proposed baselines are also applicable to both online and offline class-IL.

## 2.3 Intra- and Cross-task training

### 2.3.1 Notation

We assume that the data comes in the form of the following task sequence:

$$\mathscr{T} = \{(C^1, D^1), (C^2, D^2), ...(C^n, D^n)\}, \tag{2.1}$$

where $n$ is the number of tasks, $C^t$ is the set containing the classes of task $t$, with the constraint of non-overlapping classes between different tasks ($\forall\ 1 \leq i \leq n$, $i \neq j$, $C^i \cap C^j = \varnothing$). Let $D^t$ be the dataset for task $t$, containing the labelled images. The learner is a neural network, function of its parameters $\theta$ and the input $x$, which we will denote by $f(x;\theta)$. We further split this network into a feature extractor $\Psi(x;\theta_\Psi)$ with weights $\theta_\Psi$, and a classification layer $\nu(x;\theta_\nu)$ with weights $\theta_\nu$. The logits are obtained by applying the feature extractor first, then the classification layer, obtaining $f(x;\theta) = \nu(\Psi(x;\theta_\Psi);\theta_\nu)$. The upper-scripted $\theta^t$ denote the weights of the model after seeing task t. Additionally, we denote the current task as $T$, a batch of data as $\mathscr{B}$, and

the upper-scripted subset containing data from task $t$ as $\mathscr{B}^t = \{(x, y) \in \mathscr{B}^t, y \in C^t\}$. So that we can consider all classes and data that the network has seen so far, we define $C^k_\Sigma = \bigcup_{t=1}^k C^t$ and $D^k_\Sigma = \bigcup_{t=1}^k D^t$, where $\Sigma$ stands for cumulative.

### 2.3.2 Two Replay Baselines

In order to better dissect the replay mechanism, we propose two different baselines to address class-IL. The two methods differ only in the loss they apply to the feature extractor. The first one explicitly tries to learn both cross-task and intra-task discriminative features, while the second one avoids learning cross-task discriminative features, restricting itself to learn only intra-task ones. They are defined as:

$$\mathscr{L}_{CE}(\mathscr{B};\theta) = \sum_{(x,y)\in\mathscr{B}} -\log \frac{\exp f(x;\theta)_y}{\sum_{c\in C^T_\Sigma} \exp f(x;\theta)_c}, \tag{2.2}$$

$$\mathscr{L}_{CE\text{-}IT}(\mathscr{B};\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{(x,y)\in\mathscr{B}^t} -\log \frac{\exp f(x;\theta)_y}{\sum_{c\in C^t} \exp f(x;\theta)_c}. \tag{2.3}$$

$\mathscr{L}_{CE}$ (Eq. 2.2) is the classical cross-entropy over all classes seen so far, which seems natural to use whenever using exemplars. $\mathscr{L}_{CE\text{-}IT}$ (Eq. 2.3) is the sum of the cross-entropies of each separate tasks, as it would be done in multi-task training or task-incremental learning, this loss is only learning intra-task discriminative features ($CE\text{-}IT$ stands for cross-entropy intra-task). During training, $\mathscr{B}$ will be drawn from $D^t \cup \mathscr{M}$ where $\mathscr{M}$ is a memory buffer that retains a small number of samples per class.

To tackle task-recency bias that might arise from unbalanced training, and calibrate the classification heads on top of the feature extractor, we add an extra step to our proposed baseline algorithms. Taking inspiration from EEIL [27], we perform an additional balanced fine-tuning step after learning the current task, training the network on data only from the exemplars memory buffer, which contains a balanced number of training samples per class. We use $\mathscr{L}_{CE}$ on both cases for this step, but only back-propagating it through the classifier, leaving the feature extractor frozen with the previously learned knowledge. This is a similar procedure as $FT^{BAL}$ [17] with the difference of the feature extractor parameters being frozen while the balancing step takes place.

The detailed training procedure is explained in Algorithm 1, where $\mathscr{L}$ is the loss used during the first training step, and is replaced by $\mathscr{L}_{CE}$ or $\mathscr{L}_{CE\text{-}IT}$ depending on the baseline we are using. We will later refer to these baselines as $FT^{BAL}_{+CTF}$ (using $\mathscr{L}_{CE}$) and $FT^{BAL}_{-CTF}$ (using $\mathscr{L}_{CE\text{-}IT}$) where $+CTF$ and $-CTF$ stand for with and without Cross

**Algorithm 1:** Two step procedure

**Input:** task sequence $\mathscr{T}$, data $D$, mem. buffer $\mathscr{M}$, loss $\mathscr{L}$
**for** $t \in 1 \dots T$ **do**
  **for** $i \in 1 \dots N_{\text{epochs}}$ **do**
    **for** $\mathscr{B} \sim D^t \cup \mathscr{M}$ **do**
      $\theta' \leftarrow \text{SGD}(\mathscr{L}, \mathscr{B}, \theta)$
    **end for**
  **end for**
  $\mathscr{M} \leftarrow \text{FillMemory}(D^t, \mathscr{M})$
  **for** $i \in 1 \dots N_{\text{FTepochs}}$ **do**
    **for** $\mathscr{B}_{mem} \sim \mathscr{M}$ **do**
      $\theta''_v \leftarrow \text{SGD}(\mathscr{L}_{CE}, \mathscr{B}_{mem}, \theta')$
    **end for**
  **end for**
**end for**

Task Features respectively. Indeed, $FT^{BAL}_{\text{-}CTF}$ will emulate the learning of a feature extractor as it would be done in the task-incremental learning setting. This will allow us to evaluate how features learned similarly as in the task-IL setting perform when used in the class-IL scenario, without access to task id. This fine-tuning step can be seen as a sort of probing of the feature extractor, similarly to what is done in [4], though here it is performed on a reduced dataset.

Naturally, we expect $FT^{BAL}_{+CTF}$ to perform better in the class-IL setting since it is targeted to that setting. Our interest will rather focus on the gap between $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{\text{-}CTF}$. Studying this gap and its contribution to the larger gap between these methods and the upper bound for this setting will help us understand how improvements in task-incremental learning can benefit class-incremental learning.

Finally, we shortly recall the main challenges for task-IL since these are inherited by class-IL. In task-IL, since the task-ID is available at test time, it is not necessary to learn any cross-task features to obtain good performance on each specific tasks. This setting stills brings its share of challenges. Catastrophic forgetting [59, 82] is one of the challenges that has been addressed the most in the literature. In task-IL, it is described as the drop in performance on previous tasks when the learning of a new task occurs. While catastrophic forgetting can be tough to solve in some settings and without access to old samples, when a memory buffer of reasonable size is used (as it is done in class-incremental learning replay methods), it becomes easier to avoid and many works have shown how to prevent it [30, 104]. However, other challenges have been identified, among them, the one of knowledge transfer between tasks. Primarily

discussed in the multi-task learning literature [14], the transfer of knowledge between tasks enables the learner to boost its performance on separate tasks when it is learning them concurrently. In the task-incremental learning setting, this transfer between tasks has been referred to as backward- and forward- transfer in lopez2017gradient [104]. Where backward denotes the positive influence on previous tasks performance that learning subsequent tasks could have, while forward denotes the positive influence learning previous tasks could have on the learning of new tasks.

### 2.3.3 On forgetting in class-incremental learning

Catastrophic forgetting [59, 82] has been widely cited as being the main cause of performance drop in many continual learning settings. From task-IL works with two tasks similar to transfer learning [82], the measurement of forgetting was defined as how much of the performance on the source task decreased when learning the target task. Initially, this measurement was very useful even when moving from two tasks to a sequence of them since each task was evaluated separately. It has been widely observed that training a neural network on a sequence of tasks without accessing data from previous tasks results in catastrophic forgetting of previously learned ones. Similar variations of that metric have also been proposed for both task-IL and class-IL [28, 91, 144]. However, we argue that they do not directly transfer to class-IL since, in that setting, the average accuracy is reported on an ever-more difficult problem, which is classifying between all $C^n$ classes. For this reason, looking at the classic forgetting measure in class-IL it is not clear if it captures only the forgetting of previous tasks but also the increasing difficulty of the cumulative task.

**Task-IL metrics:** Consider that $a_k^t \in [0,1]$ denotes the accuracy of task $k$ after learning task $t$ ($k \leq t$), then the *average accuracy* at task $t$ is defined as $A_t = \frac{1}{t}\sum_{i=1}^{t} a_i^t$. *Forgetting* estimates how much the model forgot about previously learned task $k$ at current task $t$ and is defined as $f_k^t = \max_{i\in\{k,\dots,t-1\}} a_k^i - a_k^t$. As with accuracy, this measure can be averaged over all tasks learned so far: $F^t = \frac{1}{t-1}\sum_{i=1}^{t-1} f_i^t$ [30]. We denote this version of forgetting as *classic forgetting*.

**Class-IL metrics:** To consider these metrics for the case of class-IL, we have to replace the notion of separate tasks that is used in task-IL by the cumulative task that is composed of the concatenation of all tasks seen so far $\mathscr{T}_\Sigma^k = (C_\Sigma^k, D_\Sigma^k)$. When evaluating a network on the cumulative task $k$, we propose the following method to predict the label:

$$\hat{y}_k(x;\theta) = \underset{c\in C_\Sigma^k}{\operatorname{argmax}} f(x;\theta)_c\,, \tag{2.4}$$

which restricts the output to the $C_\Sigma^k$ classes. Note that the predictions can change when

| scenario | approach | growing memory size | | | | |
|---|---|---|---|---|---|---|
| | | 20/cls | 10/cls | 5/cls | 2/cls | max |
| 10 tasks | $FT^{BAL}_{+CTF}$ | 40.55 ± 2.1 | 31.5 ± 3.0 | 19.1 ± 2.5 | 9.0 ± 0.69 | 66.8 ± 1.1 |
| | $FT^{BAL}_{-CTF}$ | 34.8 ± 2.1 | 27.4 ± 1.5 | 20.1 ± 1.2 | 14.3 ± 0.5 | 56.6 ± 1.2 |
| 20 tasks | $FT^{BAL}_{+CTF}$ | 34.2 ± 1.9 | 26.6 ± 2.7 | 12.5 ± 1.6 | 4.4 ± 0.15 | 67.0 ± 1.5 |
| | $FT^{BAL}_{-CTF}$ | 26.8 ± 2.6 | 20.1 ± 2.0 | 15.1 ± 1.7 | 9.2 ± 0.7 | 51.8 ± 2.3 |

Table 2.1: Average accuracy after learning all tasks for CIFAR-100 on ResNet-32 from scratch.

you chose another task $j \neq k$, which is desirable since it makes the task easier when $k$ gets smaller, getting rid of the confusion between classic forgetting and the growing task difficulty discussed above. We further introduce the term *cumulative accuracy* of task $k$ as:

$$b_k^t = \frac{1}{|D_\Sigma^k|} \sum_{x,y \in D_\Sigma^k} 1_{\{y\}}\left(\hat{y}_k(x;\theta^t)\right), \tag{2.5}$$

which refers to the accuracy obtained on the cumulative task $k$ by the model $\theta^t$, and where $1_{\{y\}}$ is the indicator function which is 1 when the prediction is correct, and 0 otherwise. To summarise the performance of the continual learning approach after $t$ tasks have been learned we simply report $b_t^t$, which is equivalent to the average accuracy classically reported in class-IL works. We then can analogously define *cumulative forgetting* about a previous cumulative task $k$ at current task $t$ as:

$$f_k^t = \max_{i \in \{1,\dots,t\}} b_k^i - b_k^t. \tag{2.6}$$

This measure can be averaged over all tasks learned so far: $F_\Sigma^t = \frac{1}{t-1}\sum_{i=1}^{t-1} f_i^t$. The above defined measurements are an adaptation of the ones from [30] to the setting of class-incremental learning, the main difference is that ours consider the incremental cumulative tasks instead of the separate ones.

## 2.4 Experimental Results

Our implementation is based on the FACIL framework [111]. We use the same data processing and scenario configurations. We extend it with our proposed baselines to compare with its implementation of state-of-the-art methods. Our code is available at https://github.com/AlbinSou/cross_task_cil.

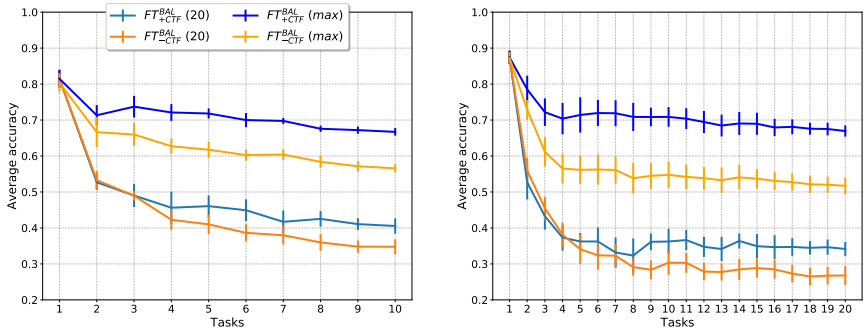


Figure 2.2: Average accuracies on CIFAR-100 splitted in 10 tasks (left) and 20 tasks (right) for $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$ using 20 exemplars per class compared to their respective upper-bounds (500 exemplars per class). Mean and standard deviation over 10 runs are reported.

**Datasets:** CIFAR-100 [85] contains 60k rgb images of size 32x32x3 divided in 100 classes, each having 500 training images and 100 test images. Since there is no defined validation set, we set apart 10% of the train set as validation, and keep it the same for all experiments. This is a common setting in class-IL [16,27,69,135,163]. The classes are divided between 10 or 20 task splits to be learned incrementally, with a memory buffer that contains 20 exemplars per class for rehearsal (2,000 total). Exemplars are selected using the herding strategy, which has been shown to be slightly more robust than other strategies [111]. Data augmentation is applied via padding, random crop and random horizontal flips during training. Finally, task and class orderings [111], are fixed to the commonly used iCaRL seed [111, 135, 163]. Imagenet [139] contains 1,000 object classes with different number of samples per class. In our experiments, we use a reduced version composed of its 100 first classes, Imagenet-Subset (as in [135]). Data is pre-processed using 224×224 random crop, normalization and random horizontal flip. We split this dataset into 25 tasks with a random class order fixed for all experiments. Exemplars are handled in the same way as the CIFAR-100 experiments.
**Network:** we use the commonly paired network for CIFAR-100 experiments: ResNet-32 [68], learned with Stochastic Gradient Descent with a patience scheme. More details about the hyper-parameters used during training can be found in Section 2.4.1. For Imagenet-Subset, we use ResNet-18, which allows for larger input size and provides more capacity.
**Upper bounds:** for each of the two baselines that use a limited amount of memory, we also consider their respective upper bounds that have access to all data from previous

tasks, namely $FT^{BAL}_{+CTF}(max)$ and $FT^{BAL}_{-CTF}(max)$.

### 2.4.1 Hyperparameter selection

We use Stochastic Gradient Descent (SGD) with momentum as the optimization algorithm, along with a learning rate scheduling strategy using patience: the learning rate decreases when the validation loss of the current task does not decrease for a defined number of epochs, the model that performs best on validation data is retained. This patience scheme is also used during the second (fine-tuning) step. Weight decay is also used for regularisation. The proposed baselines only have one additional hyperparameter, which is the number of balancing finetuning epochs from step 2. We perform GridSearch/CHF following the procedure in [40, 111] that respects the assumptions of continual learning (cannot access future tasks validation data). We perform a learning rate search by fine-tuning on the new task (without applying any other loss or strategy). Once the best learning rate is found for that task we set it as the starting learning rate for it and train the final version of the current task before moving onto the next. Below are listed the parameters used by the grid search.

Details of the Hyperparameter search for CIFAR-100

```
lr_first: [5e-1, 1e-1, 5e-2],
lr: [1e-1, 5e-2, 1e-2, 5e-3, 1e-3],
lr_searches: [3],
lr_min: 1e-4,
lr_factor: 3,
lr_patience: 10,
clipping: 10000,
momentum: 0.9,
wd: 0.0002
```

### 2.4.2 Importance of cross-task features

Results for CIFAR-100 are shown in Table 2.1. For the 10 tasks split, using $FT^{BAL}_{+CTF}$ performs better than $FT^{BAL}_{-CTF}$ when using 10 or more exemplars per class, indicating that it is able to learn some cross-task features. For very low number of exemplars, we observe that $FT^{BAL}_{+CTF}$ is performing poorly compared to $FT^{BAL}_{-CTF}$. The average accuracy gap after 10 tasks between the two methods is around ~5% when using 20 exemplars per class, which is the most commonly considered memory size for class-IL. As we move to the 20 tasks split, both methods suffer from a ~5% drop in performance. Through the comparison of these two losses with their respective upper bounds using the maximum of memory (see Fig. 2.2), we observe that the gap due to not learning

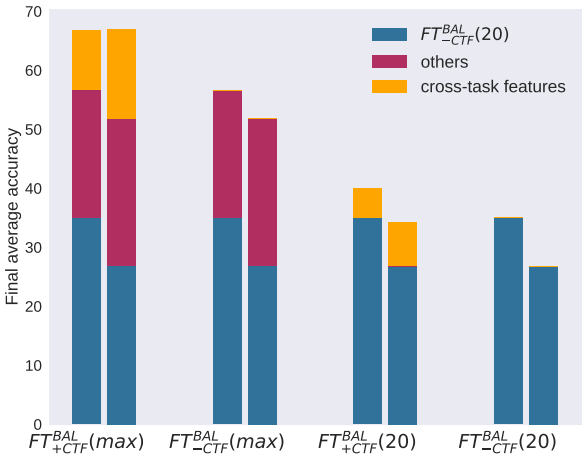


Figure 2.3: Final average accuracy obtained by each method and their respective upper bounds on CIFAR-100 splitted in 10 tasks (Left) and 20 tasks (Right). The part in red coined "others" can be obtained with better intra-task features, while the orange part is the additional gain obtained when learning cross-task features

cross-task features is of ∼10%, and jumps to ∼15% when moving to 20 tasks. This gap is filled in part (∼5% for 10 tasks and ∼7% for 20 tasks) by $FT^{BAL}_{+CTF}$ when 20 growing memory per class is used. It means that the gap due to the learning of cross-task features is already filled by a half by $FT^{BAL}_{+CTF}$, the remaining performance gap is then due to something else, materialised by the difference between the upper bound for $FT^{BAL}_{-CTF}$ $(max)$ and $FT^{BAL}_{-CTF}$ (20). We summarise these observations in Fig. 2.3.

On Imagenet-Subset (see Fig. 2.4), we observe the same conclusions. There is a consistent difference between the two baselines due to the additional learning of cross-task features. But this difference does not account for the main part of the performance gap, which is observed when comparing $FT^{BAL}_{-CTF}$ $(max)$ and $FT^{BAL}_{-CTF}$ (20), and is not related to cross-task features.

### 2.4.3 Cumulative accuracy and cumulative forgetting

We have argued that the forgetting measure defined for task-IL cannot directly be applied to class-IL (as done in some papers). Instead the definitions should be adapted to consider cumulative accuracy of the tasks (see Eq. 2.5).

We apply the metrics defined in Sec. 2.3.3 to the experiments on CIFAR-100 (10

| memory size | 10 Tasks classic | 10 Tasks cumulative | 20 Tasks classic | 20 Tasks cumulative |
|---|---|---|---|---|
| (2/class) | 81.6 | 4.0 | 86.0 | 6.4 |
| (5/class) | 69.5 | 1.6 | 74.1 | 1.1 |
| (10/class) | 56.0 | 0.9 | 49.5 | 0.7 |
| (20/class) | 28.8 | 0.4 | 39.9 | 0.6 |

Table 2.2: Average forgetting (classic) compared to newly defined cumulative forgetting measure on CIFAR100 10 and 20 tasks split for $FT^{BAL}_{+CTF}$. Cumulative forgetting shows no sign of performance drop due to a forgetting phenomenon.

Table 2.3: Final average accuracy obtained by both baselines on Imagenet-subset (25 tasks)

| memory | 20/cls | max |
|---|---|---|
| $FT^{BAL}_{+CTF}$ | 31.7 | 70.6 |
| $FT^{BAL}_{-CTF}$ | 26.4 | 62.1 |

and 20 tasks). One insightful way to analyse the learning process of class-IL methods is provided in Fig. 2.5, we coin this a *cumulative accuracy graph*. It allows you to analyse the behaviour of the various cumulative tasks while the learning progresses. For example, the line starting from Task 2 indicates $b^t_2$. There, we can observe an initial positive transfer when learning task 3 and a subsequent planar behaviour where performance remains constant. The important fact to observe from this graph is that while the average accuracy decreases considerably over the course of training, the values of the cumulative accuracy $b^t_k$ when fixing $k$ are quite stable. The resulting cumulative forgetting measure is consequently very low, and only grows for the smaller memory sizes (see Table 2.2). This means that cumulative forgetting cannot explain on its own the decrease in average accuracy. This strongly contrasts with the values obtained using classic forgetting, which are higher and indicate that most of the performance drop is caused by forgetting. In conclusion, we argue that the classic forgetting method is inapplicable to class-IL and instead encourage the usage of cumulative forgetting. Finally, for the model using maximum memory (see Fig. 2.5), we observe that the cumulative accuracies grow over the course of training, which means the model gets better at previous cumulative tasks. This is showing that positive

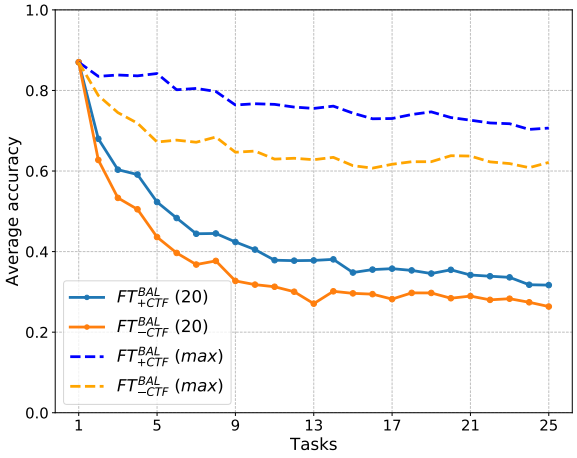


Figure 2.4: Average accuracies on Imagenet-Subset splitted in 25 tasks for $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$ using 20 exemplars per class compared to their respective upper-bounds with maximum memory. As it is the case on cifar, the gap due to the learning of cross-task features is not predominant, and is partly filled by the use of $FT^{BAL}_{+CTF}$.

backward transfer is occurring: learning new tasks improves performance on older tasks.

We applied our cumulative accuracy metric to two other state of the art methods, BiC [163] and EEIL [27], results are reported in Fig. 2.7. We observe similar results than the ones reported for our baselines. EEIL has a characteristic jump in cumulative accuracy after each task, just like our baselines. Since this method also uses a balanced fine-tuning step we believe this jump is due to the latter. On BiC, the cumulative accuracies are very stable over the course of training.

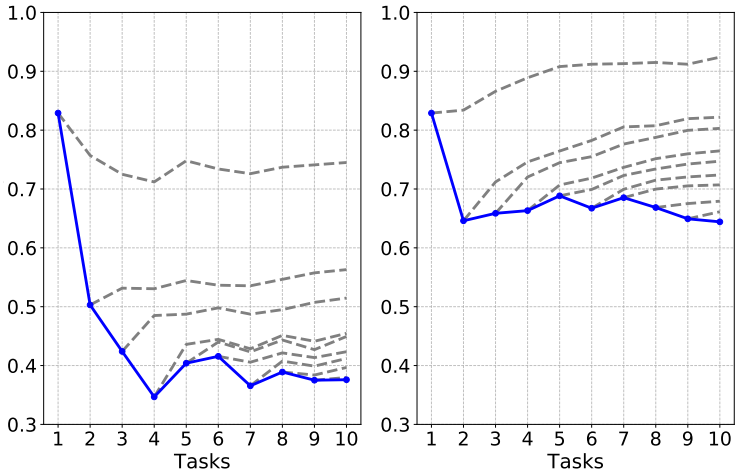


Figure 2.5: Cumulative accuracies $b_k^t$ on CIFAR100 (10 tasks) for $FT_{+CTF}^{BAL}(20mem/cls)$ (Left) and $FT_{+CTF}^{BAL}(max)$ (Right). Grey dashed lines represent $b_k^t$ for varying $t$ and one fixed $k$ per line. The blue dotted line represent $b_t^t$ for varying $t$, which is the average accuracy.

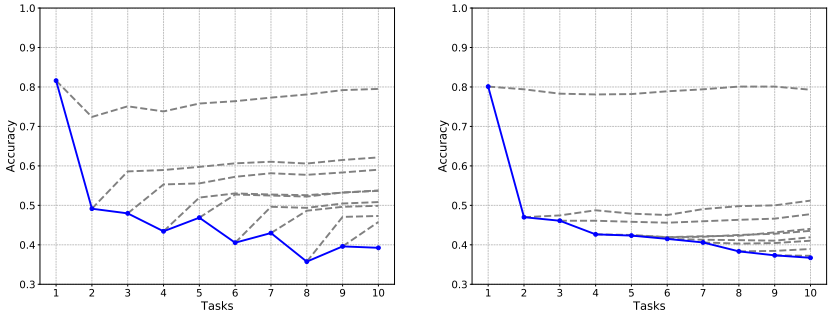


Figure 2.7: Cumulative accuracies $b_k^t$ on CIFAR100 (10 tasks) for EEIL (Left) and Bic (Right), both using a growing memory of 20 exemplars per class. Grey dashed lines represent $b_k^t$ for varying $t$ and one fixed $k$ per line. The blue dotted line represent $b_t^t$ for varying $t$, which is the average accuracy.

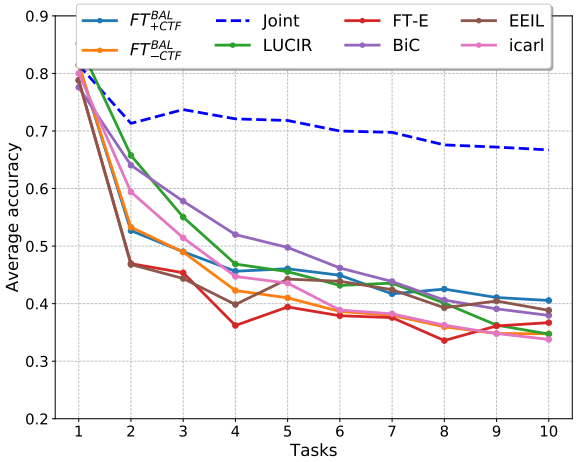


Figure 2.6: Comparison including multiple class incremental learning methods and the two baselines we use. CIFAR100 10 tasks, 20 growing memory per class. The baselines used perform comparatively to other state of the art methods.

### 2.4.4 Comparison to state-of-the-art

In Fig. 2.6, we display a comparison of the considered baselines to other state of the art methods like iCarl [135], LUCIR [69], BiC [163] and EEIL [27]. **FT-E** is another baseline which is similar to $FT^{BAL}_{+CTF}$ but does not use a second fine-tuning stage. We observe that $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$ perform comparatively to the latter methods in this setting, the comparison with **FT-E** shows the gain of the additional fine-tuning stage, which helps to get rid of the task-recency bias. The main purpose to include this comparison to state-of-the-art methods is to show that the baseline $FT^{BAL}_{+CTF}$ gets competitive results which means that our analyses on the challenges for $FT^{BAL}_{+CTF}$ could generalize to other state-of-the-art methods.

### 2.4.5 Fixed memory size

In the previous results, we considered a growing memory size. When using a fixed memory size equivalent to the final growing size we naturally obtain better results but we found them harder to interpret since in that case the number of samples available per class varies during training, which makes the incremental learning problem easier in the first few tasks and then gradually harder. Nevertheless, we report results using fixed memory sizes below (see Fig. 2.8). In this case for the first few tasks $FT^{BAL}_{+CTF}$

is able to outperform the upper bound that does not learn cross-task features, this is because enough memory is available at that time. For instance, on CIFAR100 (20 tasks) the gap between $FT^{BAL}_{+CTF}(2000)$ and $FT^{BAL}_{-CTF}(2000)$ is the same than the gap between $FT^{BAL}_{+CTF}(max)$ and $FT^{BAL}_{-CTF}(max)$ at task 3, hinting that $FT^{BAL}_{+CTF}(2000)$ was able to learn correct cross-task features until that point, as new tasks come however, less memory per class is available, rendering the learning of the latter harder.

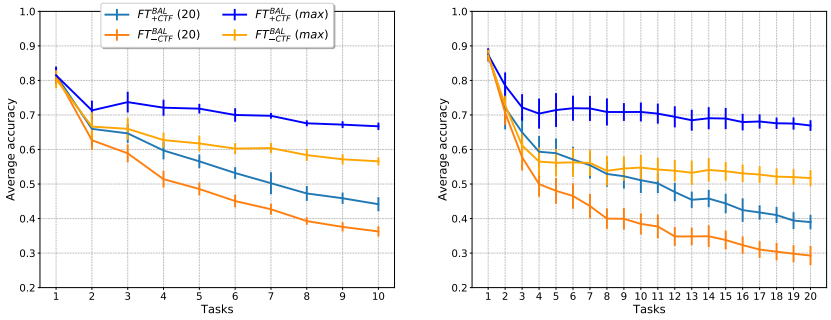


Figure 2.8: Average accuracies on CIFAR-100 splitted in 10 tasks (top) and 20 tasks (bottom) for $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$ using a fixed memory of 2000 exemplars compared to their respective upper bounds. Mean and standard deviation over 10 runs are reported.

### 2.4.6 Linear probing of intermediate layers

In this section, we aim to evaluate the quality of the representations learned by both $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$ methods in terms of linear probing accuracy [4, 38] at different levels in the network. This technique is widely used in unsupervised learning in order to check the quality of a learned representation, and while it cannot be applied in a realistic continual learning setting, because it requires access to all previous data, it is a good way to evaluate learned representations. In Figure 2.9 we plot the accuracy of linear probes learned on the output of various layers of the resnet32, and we compare the results for both $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$. We used a limited amount of memory during training (20 exemplars per class) as well as all the data (incremental joint).

We first observe that the difference between $FT^{BAL}_{+CTF}(max)$ and $FT^{BAL}_{-CTF}(max)$ is not so marked in the early layers, and more marked in the later layers (Starting from layer index 10 in the figure). We also see that this difference is created at earlier layers and is more important for checkpoints trained on CIFAR-100 splitted in 20 tasks, which makes sense since cross-task features become more important with increasing number of tasks. Another interesting observation is that while for $FT^{BAL}_{+CTF}(max)$ the

maximum accuracy is reached at the last layers, this is not the case for $FT^{BAL}_{\text{-}CTF}(max)$, where earlier layers from the last block have a better probed accuracy. This observation is consistent with the ones from [143], that claim that representations that are learned in a supervised manner are not optimal for transfer to other tasks when taken at the last layer. In this specific case, the representation learned by $FT^{BAL}_{\text{-}CTF}(max)$ is optimal for the last task of the stream (and all other tasks considering their task-specific classes separately), but not for the full task considering all classes at once.

Interestingly, for the continual methods that have learned with reduced memory, we see that in both cases (10 and 20 tasks), the features learned by the continual learning methods $FT^{BAL}_{+CTF}(20)$ and $FT^{BAL}_{\text{-}CTF}(20)$ do not differ a lot until the very last few layers. In general, both of these curves are very close until the very end and the main separation happens after the average pooling layer, both of them following a trajectory that is more similar to the one of $FT^{BAL}_{\text{-}CTF}(max)$ than to the one of $FT^{BAL}_{+CTF}(max)$ (with the best accuracy occurring at a layer that is not the last one). This shows that the network really struggles even with 20 exemplars per class in memory to learn any cross-task features in the earlier layers. The network is only able to learn some cross-task features in the very last layers.

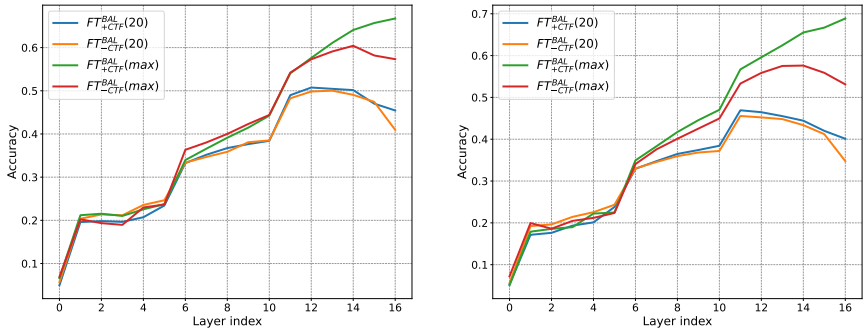


Figure 2.9: Final average accuracy after linear probing, for $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{\text{-}CTF}$ using limited amount of memory (20 exemplars per class) as well as their respective upper bound. Linear probes are learned on final model checkpoints after continually training on CIFAR-100 splitted in 10 tasks (Left) and in 20 tasks (Right).

### 2.4.7 Discussion

We here explore the efficiency of cross-task features learning done by $FT^{BAL}_{+CTF}$ as a function of the memory size. To do so, we use two additional metrics that focus on specific points of the learning. The task-aware accuracy $\mathscr{A}_{taw}$ is the accuracy of the

model on the test data when it is provided with the task-id, averaged over all tasks the model has seen so far. It is commonly used in task-IL and aims to evaluate the task-specific performance across tasks. Since class-incremental learning comes with the additional challenge of discriminating across tasks, we additionally use a task-inference accuracy $\mathscr{A}_{tinf}$. The latter is computed by counting the number of predictions that fall into the correct task. We report the values of these two measurements with an increasing memory size in Fig. 2.10.

Although the use of $FT^{BAL}_{+CTF}$, explicitly learning cross-task features accounts for a part of the gap between 20 memory and the maximum amount, we observe that both task-aware accuracy and task inference accuracy grow when increasing the memory size. Since forgetting is eliminated as a possible cause (see Tab. 2.2), the improvement in task-aware accuracy when the memory increases is most likely explained by knowledge transfer between tasks. This happens for both $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$ and indicates that there is a greater potential gain in task-inference by learning better quality task-specific features (For instance, through improved backward and forward transfer) rather than by learning better cross-task features. Indeed in this case, the learning of cross-task features is already maximum at the 50 exemplars mark (the gap between the red curve and blue curve does not increase anymore).

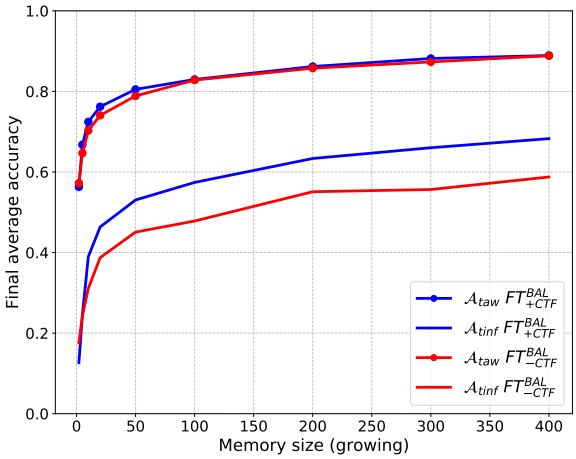


Figure 2.10: Task inference and Task aware accuracy obtained at the end of the task sequence for different memory sizes. CIFAR100 10 tasks. The gap in task inference between $FT^{BAL}_{+CTF}$ and $FT^{BAL}_{-CTF}$ stabilises after 50 exemplars per class. Additional performance gains can be obtained by increasing task-aware accuracy, which is correlated with task inference accuracy.

## 2.5 Conclusion and future directions

Through the ablation of one major component of class-incremental learning that makes it different from task-incremental learning, we attempted to link the challenges met by these two settings. By studying the impact that the absence of explicit cross-task features learning could have on popular benchmarks, we have shown that the following is already partly solved by the use of a simple replay procedure. While the learning of cross-task feature could still be improved, its influence on the final result may not be as decisive as we might think. Instead, the major part of the gap could be due to other sources. The lack of knowledge transfer studied in task-IL could be one of them. We have also highlighted that forgetting as defined for task-IL has to be carefully adapted for class-IL. Using our proposed cumulative forgetting measure, we observed that forgetting is not the main cause of performance drop in that setting.

We hope that future research can draw more links between task-IL and class-IL. While these two settings differ, we saw that they share similar challenges. In particular, we think that it could be interesting to tackle the task-IL setting using similar memory constraints as in class-IL, and aim for better knowledge transfer instead of zero forgetting. Indeed, enabling more knowledge transfer between tasks could directly be applied to class-incremental learning by learning classification heads on top of the learned feature extractor, similarly to what we did with $FT^{BAL}_{\text{-}CTF}$).

Another promising type of approaches to improve the quality of the learnt representation is learning with an unlabelled data stream concurrently to the current task, as done in [91]. That way knowledge is transferred between the data stream and the current task instead of in-between tasks. However, this requires to have access to such a data stream with nice properties w.r.t the tasks at hand.

# 3 Improving Online Continual Learning Performance and Stability with Temporal Ensembles*

## 3.1 Introduction

Model Ensembling, or the aggregation of predictions coming from different models, is a well studied and popular topic, both in the academic literature and in practical applications [44, 65, 128]. It is known to improve performance compared to using a single model. The success of ensembling methods has been found to depend on the functional diversity of the members and the efficiency of the resulting ensemble between them [58, 160]. In continual learning, the model learns the data task by task, being exposed to one task at a time. Therefore, models at different timesteps represent a functionally diverse set of models, each one locally adapted to the current task. Such functional diversity can be exploited by ensembling techniques. In Figure 3.1, we show ensembling results on two continual learning benchmarks. Here we apply a temporal ensemble of twenty models (chosen from a large number of models saved along the training trajectory). We can observe that ensembling leads to a significant performance improvement (over 40% on both datasets), and that whenever we increase the number of tasks covered by the ensemble, gains improve. Therefore, in this chapter, we investigate the use of ensembles for continual learning and provide empirical results that show their benefit for continual learning settings. Importantly, we show that when using a practical and memory efficient ensembling method similar or even better results can be obtained.

Evaluation of continually learned agents typically occurs after learning a task. Recently, another way of evaluating has been proposed, coined *continual evaluation* [22, 88] or *anytime inference* [83]. It aims to evaluate the agent's performance at any moment during learning. In this setting, Lange et al. [88] find that continual learning agents suffer from the *stability gap*, where the performance on previous tasks decreases drastically at the start of learning a new task, before returning to normal when continuing training of the new task. This behavior is problematic in many real-world applications where the agent must be applied for inference while it is learning (i.e. for financial market forecasting, or online monitoring tasks). Ensembles

*This chapter is based on an article in the Conference on Lifelong Learning Agents (CoLLAs) 2023 [148]

can provide improved stability, as they can reduce the variance of the predictions and provide a more robust prediction. By combining multiple models, the errors of one model can be compensated for by the other models in the ensemble, leading to more stable performance.

Lastly, it is known that in class incremental learning, even when using replay, the network is prone to suffer from the task-recency bias, which is a prediction bias towards classes belonging to the last task. This phenomenon has been studied and tackled in many works [16, 69, 163]. Ensembling models biased towards different tasks has the potential to reduce the bias compared to the single models; we will analyse this in this chapter.

The reasons discussed in the introduction motivate us to investigate the potential of temporal ensembles in online continual learning. Specifically, we believe that they could offer improved stability, reduce task recency bias, and benefit from more functional diversity than in i.i.d learning. By exploring the performance of ensembles of models trained on sequential data, we aim to provide insights into the benefits and limitations of using such models in online continual learning settings. Our contributions are the following

- We show that naively ensembling checkpoints from different continual learning tasks yields strong performance gains for online continual learning. In search for a practical ensembling method, we take inspiration from semi-supervised learning and apply a temporal ensembling method (i.e., an exponential moving average ensemble) as an evaluation model.
- We report the performance increase of the EMA ensemble in combination with several methods, showing a consistent increase in performance (up to 9.3% on Split-MiniImagenet). We also compute continual evaluation metrics and consistently notice a great increase in stability metrics resulting from the use of the EMA ensemble (up to 32.3% on Split-Cifar10). We further observe a reduced task-recency bias of the EMA method.

### 3.1.1 Motivational experiment

Figure 3.1 shows the comparison between ensembles of models that cover a different number of tasks. To perform this comparison, we first train a model continually using a dedicated continual learning method (ER and ER-ACE in that case), along the way, we save model checkpoints every 10 iterations, leaving about 460 model checkpoints at the end of learning the task (22 models per task in the case of Split-Cifar100 20 tasks). Then, for each number of tasks covered, we select a subset of models that cover no more than that number of tasks, and sample 20 models from that subset that we join into an ensemble for which we record the test performance. We sample 10 of

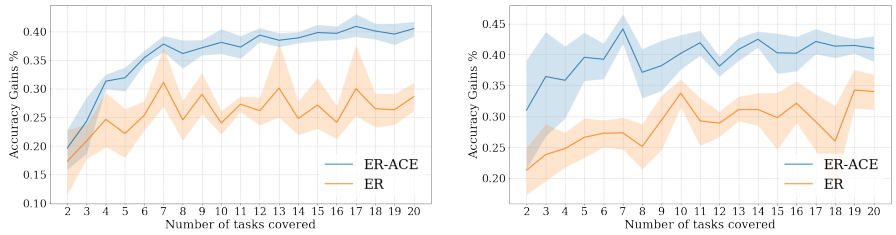


Figure 3.1: Relative accuracy gains (multiplicative in %), compared to the worst performing ensemble, when naively ensembling 20 models coming from different learning tasks on Split-Cifar100 (Left) and Split-MiniImagenet (Right) for two online continual learning methods, classical replay ER and asymmetric cross entropy loss ER-ACE. Results are reported as a function of *number of covered tasks* which is defined as the number of tasks from which the ensembled models originate. The graph shows that diversity of the ensemble is important.

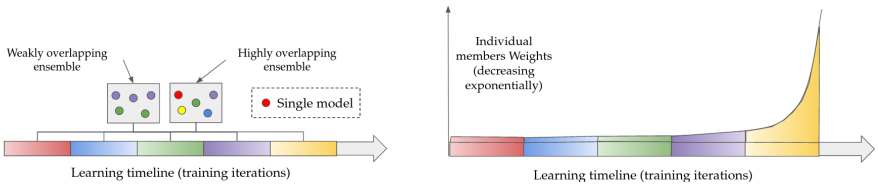


Figure 3.2: Schema of the motivational experiment ensemble (Left) and of the Exponential Moving Average ensemble (Right). For the EMA ensemble, a continuum of models is inserted in the ensemble which only occupies as much space as one additional model in the memory. Weights of previous model checkpoints decrease exponentially. This procedure permits to cover a spectrum of different tasks in online continual learning.

such ensembles for every x-axis value to get mean and confidence interval that we report in the figure. We always add a model in the ensemble which is the last training checkpoint, so that we can always give an accuracy number to later classes when sampling models. The ensembling of models coming from different tasks is done in the following way. We first compute the output probabilities that each model gives to the input. We then compute the per-class mean probability, it is possible that one class is predicted different number of times than another class by the ensemble, so we take that into account and divide the sum of the probabilities for that class by the number of models in the ensemble that can predict that class.

## 3.2 Related Work

### 3.2.1 Continual learning and online continual learning

Popular continual learning scenarios often assume that data arrive in large batches of i.i.d data, with sharp distribution shifts happening whenever a new batch becomes available. We call this setting boundary-aware continual learning, due to the additional information provided by the arrival of a new task during training. Most of the continual learning literature focus on that setting, either by also providing the task-id at test time [40] (task-incremental learning), or not [111] (class-incremental learning). We will focus on class-incremental learning in this chapter.

Boundary-free online continual learning [8] removes this form of supervision by learning on a stream of small mini-batches. This means that the granularity of the data distribution can be refined. In practice, it is possible to keep tasks that define the granularity of the distribution, while letting the agent assume this distribution could change at any new mini-batch. This is the setting that we experiment on in this chapter. In MIR [6], the authors introduce a selection strategy for replay and select the samples that will be mostly penalised by the current update. In ER-ACE [22], an asymetric cross-entropy loss is used on current data, using only the logits represented in the current mini-batch, while the classical cross-entropy loss is used on the replay buffer. [169] show that a strong baseline (called RAR) in online continual learning is repeatedly training on the available batch by sampling a new replay batch and applying data augmentations. In this chapter, we propose a simple improvement to these methods based on temporal ensembling which is applied at evaluation time.

Other, more classical class-incremental learning methods can be used in this setting, as long as they don't require knowledge of task-boundaries during training time. In ICaRL [135], a selection strategy for storing the buffer sampled is used, along with a nearest mean classifier. In DER [21], both the samples and the logits of these samples are stored and replayed using data augmentation. Other classes of approaches like EEIL [27], perform a balancing step at the end of training on each task, which makes them unsuitable to the boundary-free setting, unless the balancing is done before every evaluation session, in which case it could drastically increase the computation requirements. Other than these methods, we will focus on online continual learning methods in this chapter.

### 3.2.2 Ensembling in continual learning and temporal ensembling

Aggregating the predictions coming from multiple trained models has been known for a long time as a process that results in increased performance compared to using the predictions of the individual models [19]. The group of trained models used for

prediction is referred to as an ensemble of models. Such ensembles have been studied extensively in the literature [44, 65, 128]. Several challenges are associated with the creation of such ensembles, and a lot of the requirements of these ensembles seem at first incompatible with the constraints imposed by continual learning. In particular, naively creating an ensemble requires the training of several models that need to be stored in memory and trained independently, thus violating the memory and time constraints of continual learning. Nevertheless, some of these challenges have been already addressed in the literature. Huang et al. [72] relieve the constraint of having to train separate models by using checkpoints of the same training run and a cyclic learning rate schedule as the ensemble members. Wortsman et al. [160] train not only one network but a parametric subspace of networks which they can use to create an ensemble.

Wen et al. [157] develop BatchEnsemble, a memory efficient way to create an ensemble of models by learning a shared weight matrix for all the members, and then a rank one matrix for each of the members. A member is then computed as the result of the hadamard product between the shared matrix and the rank one matrix. They later use this technique in task incremental learning, where they learn one member per task to be used at test time. Doan et al. [46] lay the grounds of continual learning beyond the use of a single model. They study feasible ways of learning an ensemble of models continually, and compare a variety of ensembling techniques like BatchEnsemble [157] or Subspace Learning [160]. They conclude that ensembling helps in the setting of task-incremental learning, and propose a method that make use of that property to increase the performance in that setting. Compared to our work, they focus more on the effect of adding more models into the ensemble but not so much on the effect of ensembling models coming from different tasks, and they operate in the task-incremental learning setting. Lee et al. [93] propose *Incremental Moment Matching*, in which they compute the mean of the model weights in the weight space, in turn creating an approximate ensemble. In contrast to our work, they operate in the simpler task-incremental setting, in which task-ID is available at test time.

Temporal ensembling [141], is a technique that consists in ensembling the predictions coming from different models on the training trajectory. In the original work, it was done by keeping an exponential moving average of the predictions of the model on the training data, but this technique was later refined in [150], where the authors chose to keep a running average of the weights instead of the predictions, and show that this leads to similar or even better performance, while relieving the constraint of having to update the running prediction for each datapoint at every iteration. In both of these works, the resulting ensemble prediction was used to improve the results in semi-supervised learning, where only a small fraction of the sample labels are available. This same Mean-teacher model has also been used successfully in several self-supervised learning works [25, 61]. In this chapter, we study the application of

cheap temporal ensembles to the setting of online continual learning.

## 3.3 Preliminaries

### 3.3.1 Continual Evaluation and The Stability Gap

In continual classification, a learning agent learns the parameters $\theta \in \Theta$ of a function $f : (\mathscr{X}, \Theta) \mapsto \mathscr{Y}$ from the image input space $\mathscr{X}$ to the label space $\mathscr{Y}$. It does so by observing a stream of data $\mathscr{S} = \{(x^1, y^1), (x^2, y^2), ...(x^n, y^n)\}$, where $x \in \mathscr{X}$ and $y \in \mathscr{Y}$. Each data tuple is drawn from a time varying distribution $(x^t, y^t) \sim \mathscr{D}_t$. In classical machine learning the training data distribution does not depend on time, but this is added as a constraint in continual learning. In both cases the goal of the agent is to perform well on new samples drawn from the joint distribution $\mathscr{D}$, which is marginalized over past time. In practice, continual learning is simplified to allow for easier analysis by studying distributions that come from a discrete set and switch from one distribution to another (referred to as tasks $t \in \{t_1, ...t_T\}$). In this chapter, we will focus on class-incremental learning, where the learner does not have access to the task identifier $t$ at inference time.

While all of the above simplifications make sense, they are still far from the human learning experience, and from fitting the requirements of many real-world applications. In comparison to the above, humans experience continuously time-varying distributions and continual evaluation. In order to address this, Caccia et al. [22] and Lange et al. [88] lay the basis and encourage the study of continual evaluation of neural networks. In continual evaluation, the model is continuously evaluated during, instead of after each task. Interestingly, they noticed that the performance on previous tasks often drops at task shifts before coming back to a higher value later in training, this is what they refer to as the *stability gap*.

### 3.3.2 Continual Evaluation Metrics

In this section, we present various metrics used in the online continual learning setting and that can help measure the stability and more generally, evaluate the performance of the agent over the course of its training. We denote $\mathbf{A}(E_i, f_t)$ the accuracy of $f_t$ (model at current iteration $t$), on the evaluation task $E_i$. The most common metric used in this scenario is the *average anytime accuracy*, $AAA_t$ (See Eq. 3.1), used in many works [22,23,83]. While this metric does not focus on the worst-case performance, it is a nice indicator of the performance of the learning agent over the course of training. It measures the average accuracy on all tasks seen so far, and averages it over all training iterations. In [88], a set of metrics is introduced to measure worst-case performance.

These are particularly suited to assess the *stability* of the algorithms. They first define the average minimum accuracy reached by previous tasks when learning task $T_k$, min-$\text{ACC}_{\text{T}_\text{k}}$ (see Eq. 3.2). It gives a good idea of the worst case performance of the agent on a given task. Then, the worst-case accuracy, WC-$\text{ACC}_\text{t}$ (see Eq. 3.3), combines information from the minimum accuracy on previous tasks and the accuracy on the current task. WC-$\text{ACC}_\text{t}$ summarizes the trade-off between stability (accuracy on previous task data) and plasticity (accuracy on current task data). This metric is upper-bounded by the average accuracy. Here, $t$ is the current iteration, $T_k$ the current task (at iteration t) and $t_{|T_i|}$ is the iteration at the end of learning $T_i$. Since WC-$\text{ACC}_t$ is upper bounded by the average accuracy, we also report a new metric which is the relative gap between the latter and average accuracy $\text{Acc}_t$ as defined in Equation 3.4, we name it *Relative Accuracy Gap* (RAG) since this measures the relative gap between worst-case accuracy and average accuracy. This metric can then be fairly compared across various methods that have different average accuracy.

$$\text{AAA}_\text{t} = \frac{1}{t}\sum_{j=1}^{t}\frac{1}{k}\sum_{i=1}^{k}\mathbf{A}(E_i, f_j) \tag{3.1}$$

$$\text{min-ACC}_{\text{T}_\text{k}} = \frac{1}{k-1}\sum_{i}^{k-1}\min_{n}\mathbf{A}(E_i, f_n), \forall n:\ t_{|T_{i-1}|} < n \leq t \tag{3.2}$$

$$\text{WC-ACC}_\text{t} = \frac{1}{k}A(E_k, f_t) + (1-\frac{1}{k})\,\text{min-ACC}_{\text{T}_\text{k}} \tag{3.3}$$

$$\text{Acc}_\text{t} = \frac{1}{k}\sum_{i}^{k}A(E_i, f_t) \quad \text{and} \quad \text{RAG}_\text{t} = \frac{\text{Acc}_\text{t} - \text{WC-ACC}_\text{t}}{\text{Acc}_\text{t}} \tag{3.4}$$

## 3.4 Method: Exponential Moving Average ensemble (EMA)

In the introduction (see Figure 3.1) we have shown that ensembles can greatly improve performance, however, they come at a significant increase in memory usage which is in direct conflict with the memory requirements typically imposed on continual learners. Indeed, both continual learning and online learning impose memory constraints since they do not allow retaining more than a fixed amount of data coming from the stream of data. Therefore, in this section, we look into methods to reduce the memory usage, while maintaining the advantages of model ensembling.

Several works have focused on reducing the memory constraint of ensembles [150, 157, 160], some did so notably by averaging the models in weight space instead

| Method | Accuracy | Memory (Mb) |
|---|---|---|
| ER | 26.2 | 4 + 211 |
| ER Naive Ensemble | 32.1 | 1840 + 211 |
| ER + EMA | 36.3 | 8 + 211 |

Table 3.1: Comparison on Split-MiniImagenet (20 tasks) of the naive ensembling of checkpoints taken along the training trajectory of a replay method every 10 iterations, against the use of EMA ensemble. For clarity, we divide the memory footprint into the one for the models and the one for the replay buffer (model + buffer).

of aggregating the predictions in the functional space. While it is not clear under which condition such manipulation of the weights can form a model that performs similarly to the ensemble of the summed members, several works have shown practical working cases. It is possible to perform such a summation [160] whenever two models are connected by a linear path of low loss [52]. Tarvainen & Valpola [150] propose instead to do a weighted sum of an infinite amount of checkpoints by giving older checkpoints less important weight, decreasing exponentially with the distance.

For a potential use of these ensembles in continual learning we are interested in having an ensemble that covers many tasks (see Section 3.1), and one that is cheap to store and compute. The solution adopted in [150], for semi-supervised learning, fits well to this task since it requires storing only one additional model and is able to ensemble models from all of the previous iterations. Consider a function $f(x) = f(x, \theta^t)$ with learnable weights $\theta^t$ (at training iteration t), the exponential moving average (EMA) of it's weights is defined as:

$$\theta^t_{ema} = \lambda \theta^{t-1}_{ema} + (1 - \lambda)\theta^t, \tag{3.5}$$

where $\lambda$ is a user-defined hyperparameter comprised between 0 and 1, which sets the importance of the current model in the running average compared to the one of the previous models, $\theta^{t-1}_{ema}$ is the value of the moving average at the previous iteration, and $\theta^t$ is the weights of the training model at iteration $t$ ($\theta^t$ can be computed with existing online methods, such as ER [31] or MIR [6]. This implicit definition can also be written as an explicit sum over all the previous model weights:

$$\theta^t_{ema} = \sum_{i=1}^{t} (1 - \lambda)\lambda^{t-i}\theta^i + \lambda^t \theta^0_{ema}. \tag{3.6}$$

This means that the ensemble formed by the sum virtually covers all the previously

encountered tasks. However, exponentially less weights will be reserved to older tasks, potentially reducing the effective diversity of the ensemble. Nevertheless, the motivational experiment we conducted on Split-Cifar100 show that after some number of tasks covered by the ensemble, the accuracy gained by covering more and more tasks is less important (sublinear growth). So covering a small number of tasks with the ensemble can be sufficient to get satisfying performance gains. This motivated us to analyze the exponential moving average model in the setting of online continual learning.

In Table 3.1 we compare Naive Ensembling, discussed in Section 3.1, with the EMA model when combined with Experience Replay [31]. As can be seen, the EMA model significantly reduces the memory usage (requiring only one additional model). Remarkably, ER+EMA outperforms the Naive Ensemble. This could be caused by the fact that ER+EMA combines many more models, and because of the non-linear weight assignment to the various models (see Figure 3.2), where EMA assigns more weight to the last (and better) models in the training trajectory.

While the EMA approximate ensemble is commonly used in the literature and has been proven to give good performance [25, 61, 150]. Other weight summing schemes could be tried to compute an approximate ensemble. Such schemes are not necessarily expected to work in classical offline learning since summing models that are far away from each other in the weight space is not guaranteed to work. However, since online learning only performs a few training iterations on the task at hand (compared to offline learning), we can expect the models to be closer from each other and thus allow other summing techniques to work. In order to explore the possibilities of such summing techniques in online continual learning, we compare several weighting techniques. Since we are working in the online learning setting and under the constraint of storing only one additional model, we choose to compute the approximate ensemble following Equation 3.7,

$$\theta^t_{ensemble} = \frac{1}{\sum_i^t w_i} \sum_i^t w_i \theta^i, \tag{3.7}$$

and use this ensemble instead of the EMA ensemble. This formulation allows for more freedom in the choice of the weighting scheme, but the formulation used by EMA can also be expressed under this form, we identify the equivalent weight $w_i$ for the EMA ensemble using Equation 3.6 as being $w_i = \frac{w_{i-1}}{\lambda}$. We update at every iteration the weighted sum of the models and normalize it by the sum of the weights to avoid exploding model weights. In Figure 3.3 we provide a comparison of the weight distribution for the different strategies we tried.

In Table 3.2, we present the results of combining an ensemble computed with each

| | $w_i$ | ER | ER-ACE | RAR |
|---|---|---|---|---|
| - | - | 9.9 ± 0.6 | 16.5 ± 0.7 | 27.6 ± 1.3 |
| EMA $\lambda = 0.99$ | $\frac{w_{i-1}}{\lambda}$ | 14.0 ± 0.5 | 19.0 ± 0.3 | 35.4 ± 1.2 |
| EMA $\lambda = 0.995$ | $\frac{w_{i-1}}{\lambda}$ | 18.0 | **20.1** | **36.8** |
| EMA $\lambda = 0.999$ | $\frac{w_{i-1}}{\lambda}$ | **18.3** | 18.8 | 31.2 |
| Uniform | 1 | 13.1 | 12.7 | 17.3 |
| Linear | i | 16.4 | 16.3 | 25.4 |
| Logarithmic | $w_{i-1} + log(i)$ | 16.6 | 16.6 | 26.2 |
| Quadratic | $w_{i-1} + i^2$ | 18.2 | 18.8 | 31.5 |

Table 3.2: Comparison of different approximate ensembling methods inspired from the EMA approximate ensembling method. Each ensembling technique was tried on Split-Cifar100 in combination with ER [31] ER-ACE [22], and RAR [169]. The second column indicates how the weights for the current model are computed at each step. The first row indicates the results when no ensembling technique is used.

weighting scheme with three methods from the literature on the Split-Cifar100 dataset (See Section 3.5 for explanations about the experimental settings). We see that the EMA model gets the best performance overall, especially with $\lambda = 0.995$[†]. However, we get surprisingly high results with linear, logarithmic, and quadratic weighting, especially in combination with ER, but linear and logarithmic weighting fail to give an advantage when combined with the more advanced methods ER-ACE and RAR. Quadratic weighting obtains the closest resuts to the EMA methods, we provide a more detailed comparison of Quadratic against EMA method in Figure. 3.4. Uniform weighting is the scheme that performs worst across the board. which can be understood since it gives equal weight to the first models than to the last models whereas the last models have been trained for a longer time and thus are expected to have better performance.

## 3.5 Experiments

**Datasets.** We perform experiments on 3 datasets. **Cifar-10** is a 10-class dataset that contains 60000 images of size 32 by 32 and 3 color channels [85]. **Cifar-100** has the same image dimensions and number of images but with 100 classes. **Mini-Imagenet** [153] is a 100-class version of **ImageNet** [139], that contains 60000 images which are rescaled to 84 by 84. We split these datasets into 5, 20 and 20 tasks

[†]These parameters are not the one we use in the main results of Section 3.6 (we use $\lambda = 0.99$). For a discussion on hyperparameter choices, refer to Section 3.6.1

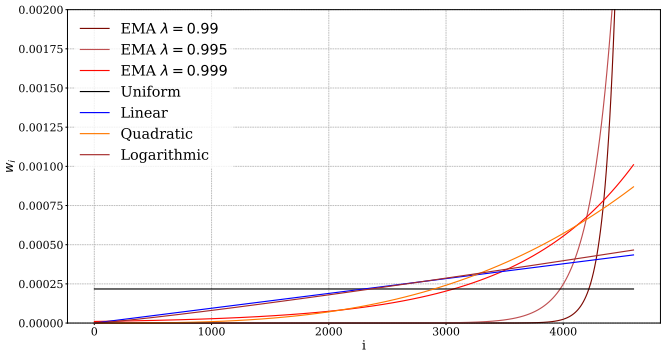


Figure 3.3: Curve showing the evolution of different weighting schemes ($w_i$) under the form described in Equation 3.7. We compare the performance of these weighting schemes in Section 3.4. We observe here that EMA with a high lambda leads to a weighting that looks similar to quadratic weighting. To draw these curves we place ourselves at the last training iteration ($\theta^{t_{last}}_{ensemble}$)

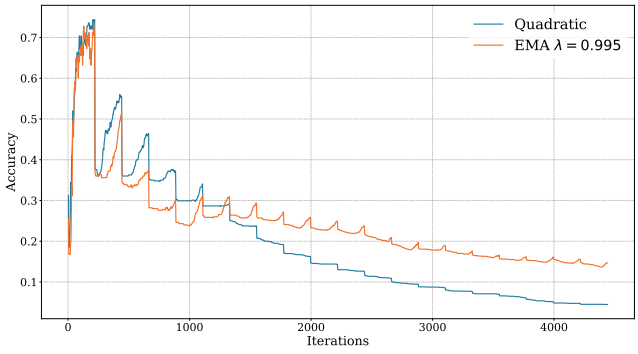


Figure 3.4: WC-ACC$_t$ for Quadratic vs EMA with $\lambda = 0.995$ in combination with ER on Split-Cifar100 dataset. We see that the quadratic weighting scheme gives more interesting stability (in terms of WC-ACC$_t$) in the first few tasks, but then fails short compared to the EMA ensemble.

respectively, each containing a mutually exclusive set of classes.

**Scenario.** We present results in the online class-incremental setting. When continual evaluation is performed, we evaluate after each unique mini-batch. All the compared methods are using a replay buffer, with a fixed memory size of 1000 exemplars for Cifar-10, 2000 exemplars for Cifar-100 and 10000 for Mini-Imagenet (as in [22]). Each training mini-batch is formed out of half of exemplars from previous tasks and half from the current task.

**Methods.** We compare the performance of five replay methods. ER-ACE [22], MIR [6], RAR [169], DER [21] are described in Section 3.2.1, while ER [31] is the vanilla replay baseline. We display their performance along with the one of their EMA augmented version on the studied datasets. Additionally, we report the results of an $i.i.d$ reference method that is allowed the same memory and computational budget as the compared methods, but for which the data arrives in an independent and identically distributed manner (in contrast to the continual learning manner where data arrives split by split). Since some of the methods included in the comparison make use of input transformations (RAR), we also include the results of the $i.i.d_{w/tr}$ reference method, which uses the same input transformations.

**Training and Implementation Details.** For all datasets we use a slim version of Resnet-18 as done in [104] and perform 3 passes per mini-batch using Stochastic Gradient Descent with a learning rate of 0.1, and batch size of 32. We run each experiment for six seeds and report the mean and standard deviation. For DER, we stick to the parameters used in the original paper for CIFAR10 ($\alpha = 0.1$ and $\beta = 0.5$). For the EMA ensemble, we chose a momentum parameter of $\lambda = 0.99$. More details on the choice of this parameter can be found in Section 3.6.1. We make use of the Avalanche framework [103] for all experiments. We make the code available at: https://github.com/AlbinSou/online_ema.

**Metrics.** For every method, we report the final average accuracy but also the continual evaluation metrics described in Section 3.3.2, that we computed on a held out validation set after training on each new mini-batch. The validation dataset contains 5% of the total training data. We report both $\text{AAA}_{\text{T}_{\text{final}}}$ and $\text{WC-ACC}_{\text{T}_{\text{final}}}$ in the tables, where $T_{final}$ is the last training iteration. We also report $\text{WC-ACC}_{\text{t}}$ at every iteration in the figures. The final value of the $RAG$ metric that we define in Equation 3.4 is reported in percent. Note that we do not make use of this validation data to tune hyperparameters but just to compute the continual evaluation metrics.

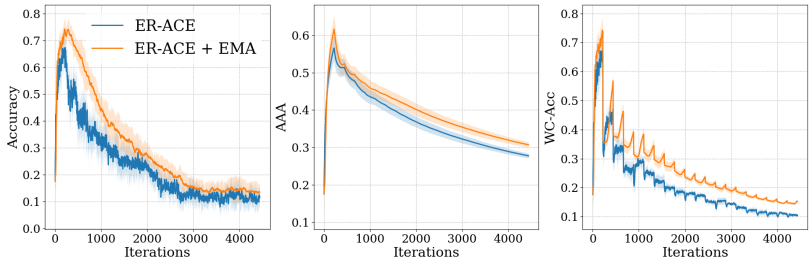


Figure 3.5: Validation accuracy on task 1 data (Left), Average Anytime Accuracy $AAA_t$ (Middle) and WC-ACC$_t$ (Right) for ER-ACE and its EMA augmented version on Split-Cifar100, using 2000 memory. Mean and standard deviation are computed over 6 runs.

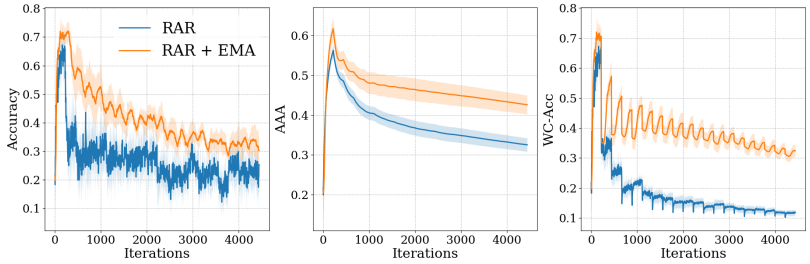


Figure 3.6: Split-Cifar100, validation accuracy on task 1 data (Left), Average Anytime Accuracy $AAA_t$ (Middle) and WC-ACC (Right), for RAR and its EMA augmented version, using 2000 memory. Mean and standard deviation are computed over 6 runs.

## 3.6 Results

On **Split-Cifar10** the EMA ensemble offers consistent improvements across all methods (see Table 3.3), especially for RAR, where it offers a 4.3% improvement in final average accuracy. We hypothesise that the gains of EMA are smaller than on Split-Cifar100 and Split-MiniImagenet because in that case the EMA ensemble weights cover less tasks than in the case of the other two datasets (5 tasks but the same number of training iterations than Split-Cifar100). Nevertheless, the stability metrics are greatly improved also for this dataset.

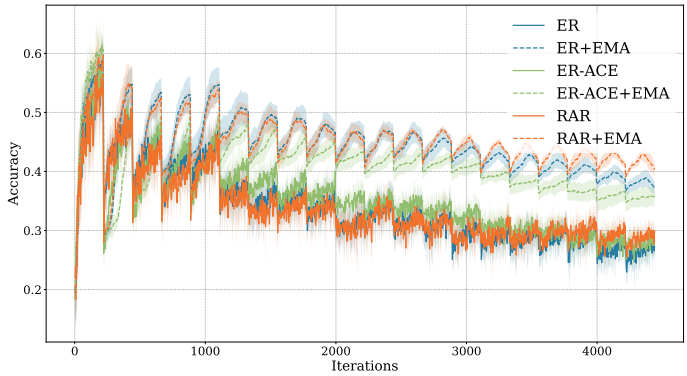


Figure 3.7: Comparison of the average accuracy on the validation data of three methods trained on Split-MiniImagenet (20 tasks), and their EMA augmented version. EMA performance is indicated with the dotted lines. We report the mean and standard deviation over 6 runs.

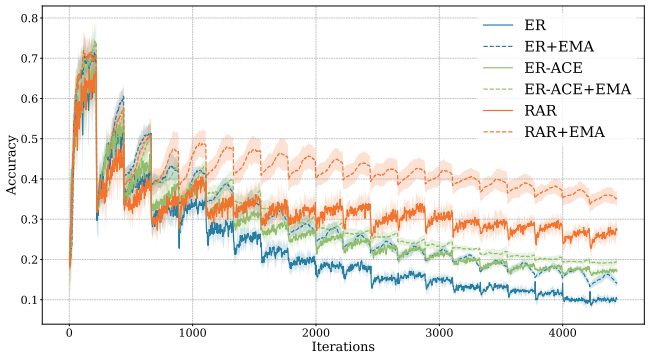


Figure 3.8: Comparison of the average accuracy on the validation data of three methods trained on Split-Cifar100 (20 tasks), and their EMA augmented version. EMA performance is indicated with the dotted lines. We report the Mean and standard deviation over 6 runs

| Method | Split-Cifar10 Acc ↑ | $AAA^{val}$ ↑ | WC-Acc$^{val}$↑ | $RAG^{val}$↓ | Split-Cifar100 Acc ↑ | $AAA^{val}$↑ | WC-Acc$^{val}$↑ | $RAG^{val}$↓ |
|---|---|---|---|---|---|---|---|---|
| $i.i.d$ | 65.2 ± 1.8 | | | | 23.4 ± 0.7 | | | |
| $i.i.d^{+EMA}$ | 67.3 ± 1.3 | | | | 25.8 ± 0.6 | | | |
| | +2.1 | | | | +2.4 | | | |
| $i.i.d_{w/tr}$ | 71.2 ± 1.4 | | | | 32.3 ± 1.0 | | | |
| $i.i.d_{w/tr}^{+EMA}$ | 75.5 ± 1.3 | | | | 37.2 ± 0.8 | | | |
| | +4.3 | | | | +4.9 | | | |
| $ER$ | 37.5 ± 1.6 | 57.3 ± 0.6 | 26.0 ± 1.0 | 31.0 ± 5.6 | 9.9 ± 0.6 | 22.5 ± 0.8 | 5.8 ± 0.4 | 43.0 ± 3.4 |
| $ER^{+EMA}$ | 38.8 ± 1.3 | 59.9 ± 0.7 | 35.1 ± 1.1 | 9.9 ± 1.5 | 14.0 ± 0.5 | 29.2 ± 0.9 | 13.1 ± 0.7 | 5.6 ± 1.6 |
| | +1.2 | +2.6 | +9.1 | -21.1 | +4.1 | +6.7 | +7.3 | -37.4 |
| $MIR$ | 40.2 ± 2.8 | 54.0 ± 0.4 | 17.0 ± 0.6 | 57.1 ± 4.4 | 10.6 ± 0.7 | 22.8 ± 0.7 | 6.3 ± 0.4 | 39.6 ± 3.0 |
| $MIR^{+EMA}$ | 42.7 ± 2.1 | 55.9 ± 4.2 | 34.5 ± 1.5 | 19.7 ± 4.8 | 14.9 ± 0.4 | 28.8 ± 0.9 | 14.3 ± 0.6 | 5.4 ± 1.4 |
| | +2.5 | +1.9 | +7.5 | -37.4 | +4.3 | +6.0 | +8.0 | -34.2 |
| $ER\text{-}ACE$ | 50.2 ± 1.2 | 62.7 ± 1.4 | 34.6 ± 2.0 | 29.9 ± 4.8 | 16.5 ± 0.7 | 27.7 ± 0.6 | 10.3 ± 0.4 | 38.7 ± 3.3 |
| $ER\text{-}ACE^{+EMA}$ | 51.5 ± 1.6 | 64.6 ± 1.4 | 48.7 ± 1.9 | 4.6 ± 0.8 | 19.0 ± 0.3 | 30.7 ± 0.6 | 15.3 ± 0.6 | 20.9 ± 3.5 |
| | +1.3 | +1.9 | +14.1 | -25.2 | +2.5 | +3.0 | +5.0 | -17.8 |
| $DER_{++}$ | 51.1 ± 3.0 | 56.4 ± 1.6 | 17.9 ± 0.6 | 64.5 ± 2.9 | 18.2 ± 0.7 | 24.7 ± 1.0 | 4.5 ± 0.6 | 75.8 ± 3.2 |
| $DER_{++}^{+EMA}$ | 53.1 ± 3.7 | 59.5 ± 1.7 | 36.4 ± 1.1 | 30.6 ± 3.5 | 23.2 ± 1.2 | 35.0 ± 1.3 | 19.7 ± 0.9 | 16.7 ± 2.3 |
| | +2.0 | +3.2 | +18.5 | -33.9 | +5 | +10.3 | +15.2 | -59.1 |
| $RAR$ | 63.0 ± 1.4 | 67.3 ± 0.9 | 28.2 ± 2.9 | 55.1 ± 4.1 | 27.6 ± 1.3 | 32.5 ± 1.5 | 11.8 ± 0.9 | 57.0 ± 1.8 |
| $RAR^{+EMA}$ | **67.3 ± 0.8** | **72.4 ± 0.8** | **60.5 ± 1.1** | 10.1 ± 1.5 | **35.4 ± 1.2** | **42.6 ± 2.1** | **32.4 ± 1.7** | 7.9 ± 0.9 |
| | +4.3 | +4.9 | +32.3 | -40.0 | +7.8 | +10.1 | +20.6 | -49.9 |

Table 3.3: Comparison of the final average accuracy on the test set and continual evaluation metrics on the validation set for various methods with their EMA model augmented version, on Split Cifar10 (Left) and Split Cifar100 (Right).

On **Split-Cifar100** (See Table 3.3 and Figure 3.8) the EMA ensemble offers considerable improvements for ER, RAR, DER and MIR (from 4.0-7.8%). The improvements are less consequent for ER-ACE, but still important (2.5%). We hypotesize that the smaller gain is due to the smaller task-recency bias of ER-ACE. This is illustrated in Figure 3.5 and Figure 3.6. In these two figures on the left, we see that the performance of RAR on task 1 data drops instantly after learning task 1, which means it has been traded for accuracy on task 2, it suffers from the task-recency bias. Whereas for ER-ACE, the performance on task 1 does not drop as much after learning task 1, thus the gap with the EMA augmented version is not as important. This is also reflected in Table 3.3. In these figures as well, we can see how the use of the EMA model improves the stability, both by looking at the WC-Acc but also at the reduced accuracy variations on a single task.

On **Split-MiniImagenet**, we see the largest performance improvements. This time, RAR sees similar gains than ER (9.3% and 10%), while ER-ACE gains are also more important than in the case of Split-Cifar100. This is coherent with the observations that we had in the motivational experiments (Figure 3.1) that the gains from ensembling are slightly more important in the case of Split-MiniImagenet. This might also be

| Method | Split-MiniImagenet Acc ↑ | $AAA^{val}$ ↑ | WC-Acc$^{val}$ ↑ | $RAG^{val}$ ↓ |
|---|---|---|---|---|
| $i.i.d$ | 29.9 ± 2.2 | | | |
| $i.i.d^{+EMA}$ | 34.9 ± 1.6 | | | |
| | +5.0 | | | |
| $i.i.d_{w/tr}$ | 32.3 ± 1.7 | | | |
| $i.i.d^{+EMA}_{w/tr}$ | 39.9 ± 1.1 | | | |
| | +7.6 | | | |
| $ER$ | 26.2 ± 0.2 | 33.9 ± 0.7 | 11.0 ± 0.9 | 59.6 ± 2.2 |
| $ER^{+EMA}$ | 36.3 ± 1.1 | 44.3 ± 0.9 | 34.2 ± 0.6 | 7.9 ± 1.4 |
| | +10.1 | +10.4 | +13.2 | -51 |
| $MIR$ | 27.3 ± 1.7 | 33.9 ± 0.4 | 9.6 ± 0.9 | 66.3 ± 2.3 |
| $MIR^{+EMA}$ | 36.1 ± 1.2 | 43.5 ± 0.8 | 34.0 ± 1.4 | 8.6 ± 2.0 |
| | +8.8 | +9.6 | +24.4 | -57.7 |
| $ER\text{-}ACE$ | 27.4 ± 1.7 | 35.3 ± 0.5 | 12.9 ± 0.9 | 54.2 ± 2.6 |
| $ER\text{-}ACE^{+EMA}$ | 34.5 ± 0.8 | 41.0 ± 0.8 | 19.8 ± 1.1 | 44.6 ± 1.8 |
| | +7.0 | +5.7 | +6.9 | -9.6 |
| $DER_{++}$ | 18.4 ± 1.8 | 20.3 ± 2.6 | 4.0 ± 0.4 | 78.5 ± 1.5 |
| $DER^{+EMA}_{++}$ | 23.3 ± 2.3 | 29.4 ± 2.9 | 17.8 ± 3.3 | 25.0 ± 6.9 |
| | +4.9 | +9.1 | +13.8 | -53.5 |
| $RAR$ | 29.1 ± 0.8 | 33.8 ± 0.8 | 11.4 ± 0.5 | 61.6 ± 2.1 |
| $RAR^{+EMA}$ | **38.4 ± 0.8** | **44.9 ± 0.4** | **35.6 ± 0.4** | 11.6 ± 1.7 |
| | +9.3 | +11.1 | +24.2 | -60.0 |

Table 3.4: Comparison of the final average accuracy on the test set and continual evaluation metrics on the validation set for various methods along with their EMA model augmented version, on Split-MiniImagenet.

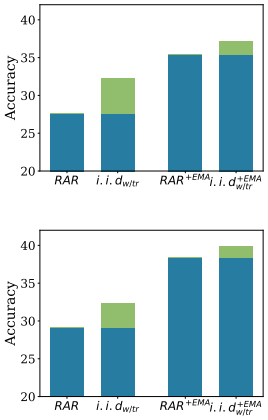


Figure 3.9: Comparison of previous state-of-the-art method in online continual learning $RAR$ against the reference method $i.i.d_{w/tr}$ on Split-Cifar100 (Top) using 2000 memory and Split-Minimnet using 10000 memory (Bottom). The performance gap is indicated in green, and is greatly reduced by the use of EMA.

due to the memory size used that is different from the one used for Split-Cifar100. In Figure 3.7, we show a comparison of three methods and their EMA augmented version. We notice that in that case, the use of ER-ACE hurts the performance of the EMA model, which performs worse than just using ER and EMA. Also, for ER and RAR, we notice various bumps in the validation accuracy [‡] of the EMA model that are not present in the current model accuracy. We believe these bumps occur when the previous task bias is compensated by bias towards the current task. The location and width of these bumps depend on the $\lambda$ parameter chosen for the exponential moving average (for more analysis see Figure 3.12).

[‡]More generally we observe these bumps on all the studied datasets and report them for Split-MiniImagenet and Split-Cifar100 in Figure 3.7 and Figure 3.12

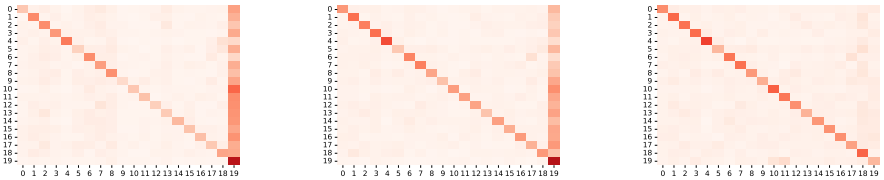

Figure 3.10: Task Confusion matrices computed on the test set after training of the last task for RAR on Split-MiniImagenet (20 tasks), final training model (Left), RAR+EMA model (Middle), and RAR+EMA model taken at the tip of the bumps observed in Figure 3.7 (Right). Note the drop in task-recency bias from RAR (a) to RAR+EMA (b) as a consequence of ensembling.

**Effect on the task-recency bias:** In Figure 3.10, we display the task confusion matrices of RAR, for the final training model, the final EMA model, and the EMA model taken at the tip of the bump (selected using a hold-out validation memory mechanism). The matrix shows the number of test images from a particular task (y-axis) that are classified as being from another task (x-axis). For the last training model, a lot of samples are predicted to be in the last task as indicated by the last column, which shows an important task-recency bias. For the final EMA model, the task-recency bias is also present though slightly diminished, but for the best selected EMA model, it is almost absent, confirming our hypothesis about the origin of the bump[§].

**Comparison with the gains in online i.i.d setting:** When applied on the reference methods $i.i.d$ and $i.i.d_{w/tr}$, the use of EMA model at evaluation also improves the results by a good margin, showing that the gains obtained in continual learning are not only due to continual-learning based improvements (like reducing the task-recency bias), but also on more general online-learning improvements. However, we observe in general higher gains in continual learning than in the i.i.d setting, except on Split-Cifar10, where they are equivalent, showing that the adaptation of the momentum parameter to the distribution drift speed is essential to get continual learning related gains. To illustrate the higher gains in continual learning, we highlight the gap in final accuracy between previous state-of-the-art method (RAR), and the $i.i.d_{w/tr}$ baseline, along with the EMA augmented versions (See Figure 3.9). We see that for two of the studied datasets, the gap between $RAR$ and $i.i.d_{w/tr}$ is reduced by the use of the EMA model. For Split-Cifar100, an initial 4.7% gap is reduced to a 1.8% gap, while

[§]Note that all our results are based on the finishing point of training, which does typically not coincide with the bump.

for Split-MiniImagenet, an initial 3.2% gap is reduced to a 1.5% gap.

**Effect on the stability metrics:** Finally, for all methods and on all datasets, the *AAA* and the WC-Acc are greatly improved by the use of EMA, which shows that aside from raising the accuracy, the EMA model offers an important stability boost. We illustrate this effect by showing one single task accuracy curve along with the WC-Acc curve during training in Figure 3.5 and Figure 3.6. We see that in both cases both the fluctuations due to small-batch training and the bigger fluctuations due to task shift are reduced by the use of the EMA ensemble. We also present a detailed analysis of stability at the level of a single task shift in Figure 3.11. In general, the biggest increase in WC-Acc also correspond to the biggest decrease in *Relative Accuracy Gap* (*RAG*), and is significant, confirming that the increase in WC-Acc is not due to an increase in average accuracy, but due to a better stability.

**A closer look at the stability gap:** In Figure 3.11, we compare the accuracy of RAR [169] and RAR+EMA on the data of two subsequent tasks during the task shift. We see that the current training model instantly loses accuracy on previous task at the task shift before regaining part of this accuracy later in training. This results in an important *stability gap* since the accuracy on previous task reaches a low point during training before going back to "normal". This observation is coherent with the one made in [88]. The orange curves indicate the performance of the EMA model on the same tasks. We see that the EMA models takes longer to get good performance on new task but ends up getting better performance than the training model. EMA models also displays improved stability (performance on previous task is smoothly going from initial to final performance).

### 3.6.1 Details about hyperparameter choice for EMA model

**Hyperparameter choice.** In all of our experiments, we chose a $\lambda$ of 0.99 for the EMA model, and additionally warm up the EMA model in the first few iterations by setting this parameter to 0.9 in the beginning. This parameter tells us about the horizon of the EMA model. Since we were interested in an ensemble that covers several experiences, we chose an horizon parameter that is high enough so that the EMA model gives non-negligible weights to previous tasks. However, this process has a flaw since it requires to know how many iterations the learner is going to perform on a given task, which is not suppose to happen in continual learning or in online learning. Nevertheless, we believe more practical solutions can be applied to tune this parameter but we did not investigate them. We think the choice of this parameter should depend on the speed at which the input distribution changes, which might be captured by some

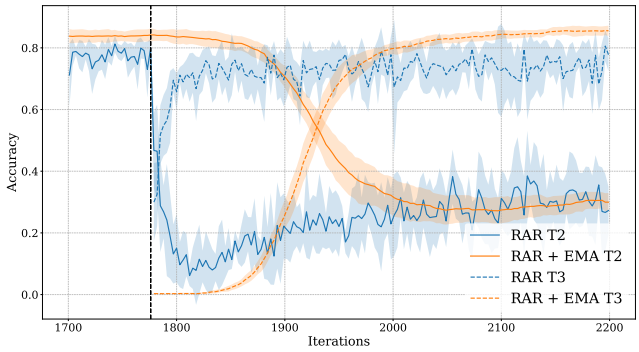


Figure 3.11: Split-Cifar10, validation accuracy on task 2 data and task 3 data at task shift (indicated with dotted black line). In blue, the accuracy for the training model, in orange, the accuracy for the EMA model. Solid lines indicate accuracy on task 2 data while dotted lines indicate accuracy on task 3 data.

other mechanism, we leave this direction to future works. In our experiments, we saw that a relatively wide range of parameters between 0.95 and 0.995 were working correctly and always boosting the accuracy in more or less significative ways. We present results for varying $\lambda$ on Split-Cifar100 in Figure 3.12. We see in that figure that increasing $\lambda$ until the accuracy bump discussed in Section 3.6 disappears is overall beneficial for the final average accuracy, but this might not be possible to do in practice when not knowing the amount of data to be received from each class. Also, higher $\lambda$ values, while beneficial for the final average accuracy, affect negatively the accuracy for earlier tasks. One work around to that problem would be to tune $\lambda$ progressively according to the number of already seen classes vs the number of new coming classes.

**The accuracy bumps** that we observe in various figures when looking at the average accuracy indicate that the gains that we report in the tables could be bigger if we would retain the model at the tip of the bump. However, this requires the use of a hold-out validation memory and an additional copy of the model on disk. In Figure 3.10, we retain the model using the above technique of validation memory and observe that these bumps correspond to a lower task-recency bias in the saved EMA model.

**Trade-off.** This choice of lambda also highlights a problem of this method that makes it difficult to export to non-online learning. It is its dependency on the number

of training iterations. Indeed, choosing a higher $\lambda$ increases the time horizon of the method, but also has negative effects for several reasons. First of all, it is not clear how high of a $\lambda$ can be chosen before the performance of the ensemble decreases since giving too much weights to single models that are too far in the weight space from the last model might break the assumption that summing models in the weight space leads to a model with accuracy equivalent to the ensemble of these two models [160]. This assumption is believed to be true as long as two models are connected by a linear path of low loss, and it is possible that two models that are far away from each other do not respect this constraint. Secondly, it would be a nice addition to be able to sum models obtained after each training batch, or even every training task (instead of each training iteration, since several iterations can be performed on the same training batch in online learning). However for the same reason invoking linear connectivity properties, it is not clear that this would work any better than increasing the momentum value in the computation of the exponential moving average.

**Task Covering.** For the chosen parameter value, on Split-Cifar100, we can compute the total amount of weight assigned to each task. From Eq. 3.6, we deduce the weight for a single member in the ensemble $\theta^i$ is $(1-\lambda)\lambda^{t-i}$ where $t$ is the iteration of the current training model. We can sum this over the iterations covering one task. If we place ourselves at the end of one task, with $\lambda = 0.99$ (value used in our experiments) we get 88% of the weights in the last task, 9% of the weight in the previous task, and 1% in the second last task. This indicates that a majority of the weights are assigned to later tasks (in particular from last task to second last task). However, we argue here that as the mean can be highly influenced by outliers, the exponential moving average is nothing else than a weighted mean, and can be influenced by outliers as well. Since the distance in the weight space between models from different tasks is more important than the one between models from the same task, it is possible that earlier models influence more strongly the current moving average than what we could believe by looking at the weights value.

### 3.6.2 Attempts at using EMA model for distillation

Tarvainen & Valpola [150] initially introduced the EMA model in order to use it as a teacher model in semi-supervised learning. They find that distilling knowledge from the EMA model to the student is beneficial in that setting. We likewise tried to apply distillation by using the EMA model as a teacher, however we found that gains obtained by this method (when present) were not robust enough to be reported. In Table 3.5, we report the student and teacher (EMA) performance when using, and not using, Mean-Teacher distillation (See Eq. 3.8). We found that some improvements could be observed in combination with simple ER on Split-Cifar100 both in terms of

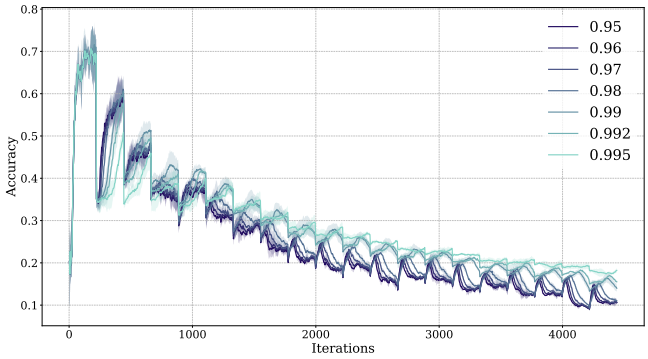


Figure 3.12: Comparison of the results obtained for various $\lambda$ values of the EMA model on Split-Cifar100 (20 tasks) for ER. Each curve depicts the average accuracy of the EMA model using a different $\lambda$ on all tasks seen so far. The value we chose in our experiments (0.99) is not optimal but nor do we have a way of chosing the optimal one. Mean and standard deviation are computed over 3 seeds.

student and teacher performance, however these improvements did not generalize to Split-MiniImagenet, neither do they combine well with a stronger method like RAR. Notably, in the case of $ER_{+MTD}$ for Split-MiniImagenet and $RAR_{+MTD}$ for both datasets, we observe that the final accuracy of the student is only slightly modified by the distllation process (sometimes increased, sometimes decreased), but the accuracy of the teacher is decreased consequently to its application, this indicates that the distillation process might reduce the diversity of the EMA ensemble by pulling models from the training trajectory closer from one another.

$$L_{CE}(f_\theta(x), y) + \alpha L_{CE}(f_\theta(x), f_{\theta_{EMA}}(x)) \tag{3.8}$$

## 3.7 Conclusion

We investigate the effect of employing temporal ensembling methods, such as EMA, in online continual learning. This direction is particularly interesting because of the distinctive nature of this combination. In online continual learning, temporal ensembles offer the potential to combine models from various training tasks, leading to novel dynamics that cannot be achieved in classical offline learning, where every model ensemble is trained on the same distribution. In the experiments, we show that

| Method | Split Cifar100 Acc | Split MiniImagenet Acc |
|---|---|---|
| $ER$ | 9.9 ± 0.6 | 26.2 ± 0.2 |
| $ER$ (EMA) | 14.0 ± 0.5 | 36.3 ± 1.1 |
| $ER_{+MTD}$ | 14.7 ± 0.5 | 27.1 ± 2.1 |
| $ER_{+MTD}$ (EMA) | 19.0 ± 0.4 | 33.5 ± 1.4 |
| $RAR$ | 27.6 ± 1.3 | 29.1 ± 0.8 |
| $RAR$ (EMA) | 35.4 ± 1.2 | 38.4 ± 0.8 |
| $RAR_{+MTD}$ | 27.0 ± 1.0 | 27.1 ± 2.5 |
| $RAR_{+MTD}$ (EMA) | 32.7 ± 0.9 | 33.4 ± 2.5 |

Table 3.5: Mean Teacher distillation results for teacher and non teacher on Split-Cifar100 (20 Tasks) (Left) and Split-MiniImagenet (Right). Mean and standard deviation are reported over 6 seeds.

temporal ensembles can greatly improve continual learning performance and stability. To circumvent the increased memory requirements for the usage of ensembles, we propose to use a memory efficient ensembling solution for online continual learning. We report results using this method in combination with other state-of-the-art methods and conclude that this method consistently increases the final performance and overall stability of several replay methods, closing in on the performance that can be reached in the $i.i.d$ setting. Most surprisingly, we do this without affecting the training process but just by ensembling models from the training trajectory.

We hope that this work inspires the design of more robust ensembling methods for continual learning. In particular, it would be important to find a similarly efficient method that allows to decorrelate the number of training iteration from the number of tasks, since it could then be applied for arbitrary number of iterations per task while similarly covering as many previous tasks as possible. Another future direction could focus on impacting the training using such kind of ensembles, for instance by combining it with distillation [¶].

[¶] We provide comments about the application of distillation in combination with EMA model in Section 3.6.2.

# 4 A Comprehensive Empirical Evaluation on Online Continual Learning*

## 4.1 Introduction

In recent years, we have witnessed a surge of interest and progress in deep continual learning methodologies. These methods are able to learn continually from a stream of non-stationary data, thereby relaxing the principal assumption of having access to *independent and identically distributed* (*i.i.d.*) samples, often made in statistical learning [66]. In classic (or batch) continual learning, the common assumption is that the data stream is composed of distinct, explicitly defined *tasks* or *domains*, and that the method can detect the task boundaries or easily switch between domains. However, in many real-world scenarios, the data stream may not have clear task boundaries or domain labels, and the method may need to quickly adapt to changes in the data distribution [9, 83]. Moreover, these methods have access to a batch of data at each task, normally in the form of a dataset, providing them with a locally i.i.d. access to the data.

*Online Continual Learning* (OCL) [104] is a more challenging and realistic setting of continual learning, where similar to online learning [117], the method learns from *each arriving data point* in the stream. OCL methods *update the model with a high frequency*, often without access to task labels or boundaries, and with a limited computational and memory budget. Due to the non-stationary nature of the stream, OCL methods need to balance stability and plasticity. Furthermore, since the model is used for inference at each time step (*anytime inference*), the learning algorithm must not suffer from stability issues at any point of the learning process.

State-of-the-art OCL methods use a rehearsal buffer to mitigate forgetting [31]. Several improvements of basic experience replay have been proposed to improve the sampling from the buffer [6, 86], the loss function [21, 22, 109], weights update [30] or the classification layer [109]. A common limitation of these works, and even recent empirical surveys [108] is that they focus only on forgetting and final accuracy. However, OCL methods have several other objectives that should be measured with

*This chapter is based on an article in the International Conference on Computer Vision Workshop on Visual Continual learning 2023 [147]

MAJOR FINDINGS OF OUR PERFORMANCE EVALUATION ON ONLINE CONTINUAL LEARNING

- Good stability does not necessarily transfer to higher accuracy (See Table 4.2, Figure 4.3 and Section 4.6 **Stability**).
- There is no best-performing OCL method across all metrics or memory sizes (See Table 4.2).
- OCL methods suffer from under-fitting in the common experimental setup (See Figure 4.3 and Section 4.6 **Forgetting**).
- Well properly tuned ER is a very competitive baseline obtaining better results than most existing methods (See **memory batch size** discussion 4.6 and Section 4.5 **Implementation**).
- The quality of the representation is very close to the one learned on the i.i.d stream, indicating that learning a good classifier is one of main problems. (See Section 4.6 **Representation quality**).

appropriate metrics.

To encourage progress in Online Continual Learning, in this chapter, we provide a *comprehensive empirical evaluation* of OCL methods. We exploit recent proposals that provide better metrics to measure forgetting (Chapter 2), continual stability [88], and quality of the representations [22, 38, 83]. The experimental results on these metrics highlight the strength and limitations of state-of-the-art methods in OCL.

In this work, we contribute to the body of literature in this area as follows:

- We formally define *Online Continual Learning* (Section 4.2) and a comprehensive set of metrics (Section 4.4) to measure the performance across different dimensions: accuracy, forgetting, continual stability, and quality of the latent representations.
- We conduct a comprehensive empirical evaluation on *Class-Incremental* scenarios on two main benchmarks (*Split-CIFAR100* and *Split-TinyImagenet*) evaluating 9 different approaches against 5 metrics. The main findings can be found in the "recommendations box" at the top of this page.
- We release all the code to reproduce our results, compare and easily prototype new OCL strategies. The code has been developed within the Deep Continual Learning Avalanche library for maximum flexibility and reproducibility.

## 4.2 Online Continual Learning

In *Online Continual Learning (OCL)*, a model learns from a stream of non-stationary experiences of data. In a classification problem, at each timestep $t$ a mini-batch $(x_t, y_t) \sim p_t(x, y)$ is available, where $p_t$ is the underlying distribution of data and may change over time. The function $f : \mathbb{R}^n \to \mathbb{R}^c$ is the prediction function and maps inputs to the unnormalized class probabilities (logits). An OCL algorithm is a function $\mathscr{A} : (x_t, y_t), f_{t-1}, \mathscr{M}_{t-1} \to f_t, \mathscr{M}_t$, where $f_t$ is the model at time $t$ and $\mathscr{M}_t = \{(x_i, y_i)\}$ is a replay buffer, *i.e.*, a small set of samples from the past stream stored for rehearsal.

Unlike Offline Continual Learning, OCL is a challenging setting where only a few new samples are available at each step, which can be stored in a very limited amount. The following are some properties that characterize OCL setups:

**Online.** At each timestep only a small mini-batch $(x_t, y_t)$ is available (10 in our experiments).

**Task Labels.** In a task-aware setting, the models know that samples belong to a set of known tasks and a task label is available to associate each sample to its own task. We assume a task-agnostic setting where task labels are not available.

**Task Boundaries.** Even in the absence of task labels, many continual learning methods assume knowledge about task boundaries, expecting to know when the data distribution switches to a new task. In a boundary-agnostic setting, this information is not available[†]. In this chapter, we test both boundary-agnostic and boundary-aware methods.

**Anytime Inference.** In most OCL applications, models should be able to train but also to perform inference online, after every training step [83]. As a result, methods that require expensive finetuning steps before inference such as GDumb [131] are not considered OCL methods in this chapter.

## 4.3 Methods

Similar to Offline Continual Learning, in OCL, most state-of-the-art results are obtained by rehearsal-based methods. Because of this, most approaches use rehearsal,

[†]This is commonly referred to as task-free, and it is a common assumption in OCL. We propose the term boundary-agnostic to highlight that task labels and boundaries are two different kinds of knowledge about the properties of the stream

| Name | Elements | Year | Boundary |
|---|---|---|---|
| AGEM [30] | Modified Update | 2018 | |
| ER [31] | - | 2019 | |
| ER + LwF [96] | Distillation Loss | 2019 | ✓ |
| ER-ACE [22] | Modified Cross-Entropy Loss | 2021 | |
| MIR [6] | Modified sampling | 2019 | |
| SCR [109] | Contrastive Loss, NMC | 2021 | |
| RAR [86] | Adversarial Augmentations | 2022 | |
| DER++ [21] | Distillation Loss | 2020 | |
| GDumb [131] | Offline finetuning on the buffer | 2020 | ✓ |

Table 4.1: Summary of methods tried in the survey along with their particularities (release year, access to task boundaries).

which is the main reason we focus on them for our empirical study. Rehearsal-based methods keep a separate memory $\mathscr{M}$ of fixed size to store past samples, updated after each mini-batch. Most rehearsal-based approaches, and all the chosen methods, follow the pseudocode shown in Figure 4.1. In the following paragraphs, we will describe in detail each line, explaining some methods-specific additions and the main reason for selecting each approach.

**Sampling.** Usually, each new mini-batch is reused for multiple training passes, each time sampling a different mini-batch from the memory and applying stochastic augmentations to both old and new data. This is justified by the theoretical analysis in [169]. MIR [6] finds the maximally interfered samples, *i.e.*, those that maximally decrease their loss after an SGD step on the new data, and select those for rehearsal. Instead, RAR [86] generates new samples using targeted adversarial attacks that are designed to be in close proximity to the decision boundary of the classifier.

**Loss.** Most methods use a supervised loss on both new and memory data, such as the cross-entropy. ER-ACE [22] use different loss on new and old data due to the different nature of the two samples. DER++ [21] uses logits distillation, storing the logits together with the raw data in the memory. While the objective is a knowledge distillation loss, the teacher used to compute the logits depend on the time when the sample was stored. SCR [109] uses a contrastive loss. In CL settings with large batches, it was shown that contrastive losses suffer less from forgetting than supervised ones [36, 50, 106]. However, many contrastive losses require large batch sizes, big and

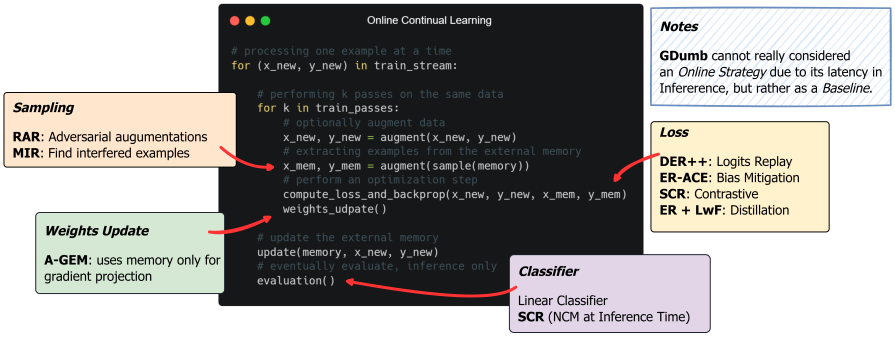


Figure 4.1: Pseudocode of replay-based OCL methods. Each method can be obtained from basic experience replay (ER) by modifying one of its fundamental components: sampling, loss, classifier, weight update.

diverse datasets, and more time to converge, which may not be possible in an online setting. Notice that distillation often requires task boundaries to know when to update the teacher.

**Classifier.** Most methods use a linear classifier trained by backpropagation. Another popular choice is the NCM (Nearest-Class-Mean) classifier, which computes a prototype for each class and uses the distance between prototypes to classify at inference time. For example, SCR uses an NCM classifier. Usually, the NCM classifier is only used during inference, while a separate linear classifier is trained via backpropagation during training.

**Model update.** Most methods use end-to-end backpropagation using both the new and the memory samples. Instead, A-GEM [30] uses the gradient from the memory to impose constraints on the update of the model, exploiting the fact that interference can be measured using the cosine similarity between gradients of different tasks.

**Baseline methods.** GDumb updates the memory via class-balanced reservoir sampling [34] and, before each inference step, retrains the entire model using only the memory data. GDumb is not an effective online method because it cannot do anytime inference due to the high cost of retraining at each step. However, it is a useful baseline that is surprisingly effective.

## 4.4 Metrics

To evaluate the performance of each strategy, we use a comprehensive set of metrics recently introduced in the OCL literature. We have taken special care to include metrics to measure stability and knowledge accumulation. Following [88], we perform continual evaluation after every timestep $t$ from the stream (after training on each mini-batch). In addition, we also evaluate the performance at task boundaries. We indicate the accuracy of a model $f$ at training iteration $t$ on evaluation task $E_i$ with $A(E_i, f_t)$, we denote $k$ the current training task index.

**Stability.** The *Worst-Case Accuracy* (WC-ACC) [88] provides a trade-off between the accuracy on iteration $t$ and the average minimum accuracy over all tasks learned before the current task $T_k$. Formally:

$$\text{WC-ACC}_t = \frac{1}{k} A(E_k, f_t) + \left(1 - \frac{1}{k}\right) \text{min-ACC}_{T_k}. \tag{4.1}$$

The metric *min-ACC* is defined in [88] with respect to the last task $T_k$ and can be computed as:

$$\text{min-ACC}_{T_k} = \frac{1}{k-1} \sum_{i=1}^{k-1} \min_{|T_k| < n \leq t} A(E_i, f_n), \tag{4.2}$$

where $|T_k|$ represents the last training iteration for task $T_k$.

**Average Anytime Accuracy.** The *Average Anytime Accuracy* [22] (*AAA*), sometimes called Area Under the Curve accuracy [83] ($A_{AUC}$), is a generalization of the average incremental accuracy [135] to the continual evaluation setting, and is defined as:

$$AAA_t = \frac{1}{t} \sum_{j=1}^{t} \frac{1}{k} \sum_{i=1}^{k} A(E_i, f_j), \tag{4.3}$$

**Average Accuracy/Forgetting.** Similarly to most of the works in CL, we report the *Average Accuracy* and the *Average Forgetting* as defined by [104]. The Average Accuracy is computed on $k$ tasks from a model at step $t$ by $AA = \frac{1}{k} \sum_{i=1}^{k} A(E_i, f_t)$. Similarly, the Average Forgetting is computed by $AF = \frac{1}{k} \sum_{i=1}^{k} A(E_i, f_{|T_i|}) - A(E_i, f_t)$.

**Cumulative Accuracy/Forgetting.** When applied to class-incremental learning, as done in this chapter, the above-described Average Forgetting metric measures

both the forgetting and the drop in performance because of the increasingly difficult classification task. In Chapter 2, we propose a forgetting measure tailored to class-incremental learning called cumulative forgetting. The *Cumulative Accuracy* for a model $f^t$ is computed on the concatenation of all evaluation tasks seen up to task $k$ ($E_\Sigma^k$), and only considering logits up to the classes seen in task $k$ ($C_\Sigma^k$). It can be computed as

$$b_k^t = \frac{1}{|E_\Sigma^k|} \sum_{x,y \in E_\Sigma^k} 1_y(\underset{c \in C_\Sigma^k}{\operatorname{argmax}} f^t(x)_c), \tag{4.4}$$

where $1_y(\hat{y})$ is the indicator function that is 1 if $y = \hat{y}$ and 0 otherwise. We then compute the *Average Cumulative Forgetting* (Chapter 2) across all tasks, which is simply computed from the Cumulative Accuracy as $F_\Sigma^t = \frac{1}{t-1} \sum_{k=1}^{t-1} \max_{i=1,\dots,t} \; b_k^i - b_k^t$.

**Representation quality.** In this setup, the model backbone is frozen and a linear classifier is trained on top of it using all the training data seen so far (linear probing) [38], we then report the accuracy obtained with this classifier (Probed Accuracy). This metric allows us to evaluate the quality of the representations computed with the incrementally learned backbone. This metric is computed only at task boundaries.

## 4.5 Experimental setup

**Benchmarks.** We present results on two continual learning classification benchmarks. **Split-Cifar100** is created from the Cifar-100 dataset [85], that contains 50 000 training images and 10 000 test images of size 32x32, equally divided into 100 classes that are refined from 20 super-classes. **Split-TinyImagenet** [1] is a reduced version of the Imagenet dataset [41], containing 100 000 training images resized to 64x64 and splitted in 200 classes. We split these datasets into 20 tasks using a random class order (task composition change for each random seed).

**Model and training details.** In all the experiments, we use a slim version of Resnet18 as done in previous works [104]. We use the SGD optimizer without momentum nor weight decay. We tune the learning rate for each method using hyper-parameter selection protocol defined later in this section. Since all of the compared methods make use of a replay buffer, we choose to follow a common protocol for learning that consists in performing several training passes on the same batch of data, using a different batch of memory. This protocol has been used in several works [6, 71, 169]. We choose to always apply input transformations since they have been proven to drastically reduce the overfitting on the buffer samples and to be efficient in combination

with several iterations per incoming batch [169]. We use a batch size of 10 for the current data for both benchmarks. Each training batch is then constituted of 10 images from current data and 10 images from the replay buffer, except for SCR which needs to sample a bigger batch from memory (of size 118). The number of training passes on each mini-batch is kept fixed for all methods (not tuned), and set to 3 on Split-Cifar100 and 9 on Split-TinyImagenet. On both of the benchmarks we use random cropping and random horizontal flip as input transformation, except for SCR which uses more advanced transformations. In the main results, we use a fixed memory size of 2000 on Split-Cifar100 and 4000 on Split-TinyImagenet. Additionally, we present results for more extreme amounts of memory (low and high) on Split-Cifar100 in Figure 4.3.

**Hyperparameter selection.** In order to make sure every method tried performs at its best, we use a hyperparameter selection mechanism. The constraints of the online setting usually do not allow for efficient hyperparameter selection. In [108], the first few tasks of the training stream are used as validation tasks, which is borrowed from the protocol defined in [30]. We also follow this protocol, using the first 4 tasks (out of 20), to optimize the hyperparameters by looking at the accuracy on the validation set. Contrary to what is done in [30], we do not allow the learner to learn offline on these 4 tasks, but rather put the learner in the setting of online learning. We use the tree-structured Parzen estimator algorithm [18] to guide the search of hyperparameters, by running 200 trials for each method. In the code (https://github.com/AlbinSou/ocl_survey) we provide the list of best hyperparameters found using this procedure for each method as well as the ranges used.

**Evaluation.** As done in [22, 83], and studied more in depth in [39]. We perform *continual evaluation*, meaning we evaluate the model on held-out validation data (5% of the whole stream) after learning on each new mini-batch. On top of this, we also perform a more classical evaluation on the test stream after training on each task.

**Methods.** We report results for all methods presented in Section 4.3. On top of that, we add an i.i.d. reference method, which uses the same methodology as the one of ER but learns on an i.i.d. stream instead of a class-incremental one.

**Implementation.** All methods were implemented using the Avalanche framework [26], an open source continual learning framework that provides the tools to implement new strategies and benchmarks in continual learning. While some methods were already present, we adapted some of the existing methods and added some new methods to the framework to conduct this study. The implementation of each method is modular, which allows reuse of common components (e.g., reservoir buffers) and highlights

the similarities and differences between methods. Despite our efforts to make the comparison as fair as possible, there are a few points on which it was hard to make every method coincide. We list them below:

- **Handling of batch normalisation statistics**: While sampling a batch from the current task and the memory, there is a choice that needs to be made when forwarding each batch to the model. The default solution adopted in Avalanche is to concatenate both batches and perform one pass on the model using the concatenated batch statistics (option 1). However, some methods were initially implemented by forwarding each batch separately, which could have a huge influence since in that case the separate outputs are created using each internal batch statistics (option 2). In general, while implementing the methods, we chose the option that was working best (ER: 1, DER++: 1, ER-ACE: 2, MIR: 2, SCR: 1, RAR: 2). Note that MIR also updates the batchnorm statistics when forwarding the bigger replay batch (from which it selects the samples to replay), which also has an influence on training that other methods do not have.

- **Memory batch size**: Initially, we wanted to fix the batch size memory using the hyperparameter validation protocol described above, so that each method could select it's adequate memory batch size. However, we found that when using a fixed memory size and doing the hyperparameter selection on only 4 tasks, a big memory batch size was always selected since it was giving more beneficial results after seeing only 4 tasks. This is due to the fact that the optimal use of the full memory size is close to always iterating on samples from the memory. Because of this, we chose to also fix the memory batch size to the same size as the one of the current batch (as done in most works). However, due to its use of a contrastive loss, SCR requires to sample a big batch from the memory, so we fixed the memory batch size to a higher number (118), which makes it behave differently than other methods.

- **Dynamic Classifier**: In continual learning, the learner is not suppose to know the total number of classes it will encounter during the training. This is why we implemented most methods using a dynamic classification layer that adds new units whenever encountering a new class. However, one method (DER++) requires to replay the logits of samples from previous classes into the new classes. The official implementation made use of a classification layer of fixed size, and we used the same in our experiments, making it different from what other methods do.

| Method | Split-Cifar100 (20 Tasks) | | | | Split-TinyImagenet (20 Tasks) | | | |
|---|---|---|---|---|---|---|---|---|
| | Acc ↑ | $AAA^{val}$ ↑ | WC-Acc$^{val}$ ↑ | Probed Acc ↑ | Acc ↑ | $AAA^{val}$ ↑ | WC-Acc$^{val}$ ↑ | Probed Acc ↑ |
| *i.i.d* | 35.3 ± 1.5 | - | - | 45.8 ± 0.6 | 26.5 ± 0.6 | - | - | 34.3 ± 0.5 |
| GDumb | 18.5 ± 0.5 | - | - | - | 13.1 ± 0.4 | - | - | - |
| AGEM | 3.1 ± 0.2 | 10.4 ± 0.6 | 2.9 ± 0.3 | 18.7 ± 0.8 | 2.6 ± 0.2 | 7.3 ± 0.5 | 2.6 ± 0.2 | 23.3 ± 0.6 |
| ER | 28.2 ± 1.2 | 36.6 ± 2.0 | 12.5 ± 0.6 | **44.9** ± 0.9 | 21.2 ± 0.6 | 33.9 ± 1.7 | 15.2 ± 0.5 | **35.6** ± 0.6 |
| ER + LwF | **30.4** ± 0.8 | 39.2 ± 2.0 | 15.3 ± 0.9 | 44.4 ± 0.8 | 22.7 ± 1.1 | 34.4 ± 2.4 | **17.0** ± 0.7 | 33.8 ± 0.9 |
| MIR | 29.4 ± 1.9 | 33.1 ± 3.2 | 11.6 ± 1.6 | 43.4 ± 0.7 | 21.3 ± 0.8 | 31.0 ± 1.8 | 15.2 ± 0.5 | 33.0 ± 0.4 |
| ER-ACE | 29.9 ± 0.6 | 38.5 ± 1.8 | 14.9 ± 0.9 | 42.4 ± 0.6 | **23.6** ± 0.7 | **35.0** ± 1.5 | 16.8 ± 0.7 | 34.2 ± 0.3 |
| DER++ | 29.3 ± 0.9 | 37.5 ± 2.5 | 13.4 ± 0.7 | 44.0 ± 0.8 | 22.9 ± 0.5 | 34.2 ± 4.0 | 16.3 ± 0.3 | 31.5 ± 0.9 |
| RAR | 28.2 ± 1.4 | 38.2 ± 1.6 | 14.9 ± 0.7 | 42.3 ± 0.9 | 15.7 ± 0.9 | 27.8 ± 2.8 | 10.1 ± 0.9 | 29.8 ± 0.9 |
| SCR | 28.3 ± 0.8 | **42.1** ± 2.1 | **20.3** ± 0.4 | 37.0 ± 0.3 | 16.9 ± 0.4 | 30.7 ± 1.5 | 12.3 ± 0.5 | 22.5 ± 0.4 |

Table 4.2: Last step results on Split-Cifar100 (20 Tasks) with 2000 memory (Left) and for Split-TinyImagenet (20 Tasks) with 4000 memory (Right). For each metric, we report the average and standard deviation over 5 seeds

## 4.6 Results

**Final Average Accuracy.** On **Split-Cifar100**, we found that the final performance of all compared methods is very similar to the one of vanilla replay (ER) (See Table 4.2). Except for the performance of AGEM, most methods performed quite competitively in the OCL setting (only around 5% away from the i.i.d. reference method), the best one being the introduced baseline combining ER and LwF, but only by a tiny margin. On **Split-TinyImagenet**, the results are similarly close (See Table 4.2), with the exception of ER-ACE performing better on that benchmark, and RAR and SCR underperforming.

**Stability (**WC-Acc **and** $AAA$**).** On **Split-Cifar100**, in terms of stability, the results vary much more between each method, and the final performance is not necessarily correlated with the stability of the method. For instance, the final accuracy for MIR is 1% above the final accuracy for SCR, however, SCR has about 9% better WC-Acc than MIR. In Figure 4.3, we illustrate the difference in stability between SCR and ER on Split-Cifar100 with 2000 memory. In this figure, the accuracy on the previous task drops significantly when shifting tasks in the case of ER, indicating low stability, this is not the case for SCR. In general, we observe that the $AAA$ is moderately correlated to the WC-Acc, even though they are not strongly linked in theory (it is possible to have low WC-Acc and high $AAA$).

**Representation quality.** Surprisingly, we find that the probed accuracy for most methods is close to the one of the i.i.d. reference method (45.8% on Split-Cifar100

and 34.3% on Split-Tinyimagenet). This suggests that the representation learned by methods on the continual stream is not much worse than the one learned on the i.i.d. stream. Therefore, the significant performance difference (in *Acc*) between incremental learning methods and the i.i.d. reference is most likely caused by a deterioration of the incrementally learned classifier. In the Appendix (See Tables 4.3 and 4.4), we provide results with 500 and 8000 memory on Split-Cifar100. Even when using 500 memory, we observe a similar conclusion, with the gap to probing augmenting only a bit (from 1% to 2%), showing that only a low amount of memory (5 per class in that case) is efficient to get a decent representation strength. In general, we find that the probed accuracy is not strongly linked with the stability metrics. In Table 4.2, we see that ER has in both cases the best Probed Accuracy, while it has lower than average stability metrics compared to the other methods.

**Forgetting.** In Figure 4.3, we display both the classical forgetting and the cumulative forgetting defined in Section 4.4 [149]. We see that while the classical forgetting indicates a high amount of forgetting that is increasing across the stream, the cumulative forgetting gives a different picture, indicating that for all methods, some backward transfer is achieved, and this remains quite constant across all tasks. Two curves exhibit slightly distinct behavior, namely the ones of SCR and MIR. These distinctive behaviors can also be observed in Figure 4.2. In the case of MIR, the average accuracy is initially low, and later increases to match the one of ER-ACE, resulting in high backward transfer. Whereas for SCR, the accuracy is initially high, but later meets the one of other methods, resulting in neutral forgetting (no forgetting, but no backward transfer). Classical forgetting however is not very relevant in the class-incremental learning setting because it increases consequently to an increase in the difficulty of the task (more and more classes to consider in the classification problem), which makes it hard to interpret, as such, we advise against its use in class-incremental learning (online or not online). In definitive, these forgetting numbers indicate that there is some backward transfer happening, we believe this is mainly due to the fact that the network is underfitted in online learning, due to the low number of training iterations, making it easy to gain additional performance on a task when training on subsequent tasks.

**Effect of the memory batch size.** As explained in Section 4.5, the memory batch size is set to the same as the current batch size in our experiments, except for SCR, which requires a higher one. We believe that this difference explains the good performance of SCR in the early training regime (see Figure 4.2) and in the high memory size setting. We perform additional experiments (See Table 4.4) where we use the same memory batch size for ER as the one of SCR. On Split-Cifar100 with 8000 memory, changing

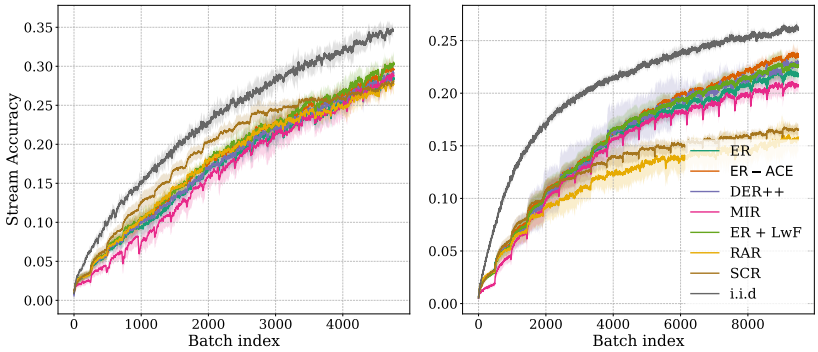


Figure 4.2: Validation stream accuracy for each of the methods, compared to the one of the i.i.d. reference method, on Split-Cifar100, using 2000 exemplars (Left), and Split-TinyImagenet, using 4000 exemplars (Right). The accuracy is reported after training on each mini-batch, we display mean and standard deviation across 5 seeds.

the memory batch size from 10 to 118 is sufficient to make ER match the performance of SCR (jumping from 34.9% to 43.0% final accuracy). This confirms our belief that this parameter is important to take into account when interpreting results.

**Effect of the memory size.** In Figure 4.3, we report the final performance of ER, the i.i.d. reference method, and the GDumb baseline when using more extreme (low and high) memory amounts on Split-Cifar100. When high amounts of memory are used, the GDumb baseline can surpass the performance of the continual learning methods if no special care is given to the memory batch size. This one needs to be adapted in order to obtain the best performances with continual learning methods. More detailed results are available in Table 4.3 and 4.4. For the low memory setting, we notice that the final accuracy results differ more between each method than when using 2000 memory, with ER-ACE getting the best results both in terms of final accuracy and stability. However, the probed accuracy is still close to the one of the i.i.d reference method. When using more memory, we see that the performance of the GDumb baselines matches the one of the i.i.d reference method. However, SCR surpasses both of these, indicating that it's still possible to learn more from the whole stream than from just the memory. We suppose that this is due to its use of a bigger memory batch size, which would be beneficial when a bigger memory size is used. To verify this, we perform an additional experiment where we provide ER with the same memory batch

| **Method** | Acc ↑ | $AAA^{val}$ ↑ | WC-Acc$^{val}$ ↑ | Probed Acc ↑ |
|---|---|---|---|---|
| $i.i.d$ | 28.3 ± 1.5 | - | - | 40.0 ± 0.9 |
| GDumb | 8.8 ± 0.5 | - | - | - |
| AGEM | 3.2 ± 0.4 | 10.4 ± 0.5 | 3.2 ± 0.3 | 19.2 ± 0.7 |
| ER | 15.7 ± 1.2 | 28.6 ± 1.7 | 7.7 ± 0.9 | **38.2** ± 1.2 |
| ER + LwF | 19.7 ± 1.5 | 32.5 ± 1.9 | 10.6 ± 0.9 | 38.0 ± 1.6 |
| MIR | 15.7 ± 1.4 | 27.4 ± 2.4 | 9.3 ± 7.7 | 36.2 ± 1.0 |
| ER-ACE | **20.8** ± 0.9 | **32.8** ± 2.2 | **11.5** ± 0.5 | 36.8 ± 1.1 |
| DER++ | 15.2 ± 1.4 | 28.9 ± 3.0 | 7.9 ± 0.6 | 37.1 ± 1.5 |
| RAR | 14.6 ± 1.2 | 28.6 ± 1.5 | 7.9 ± 0.6 | 35.7 ± 0.9 |
| SCR | 13.2 ± 0.5 | 29.4 ± 1.9 | 8.5 ± 0.5 | 28.4 ± 0.5 |

Table 4.3: Last step results on Split-Cifar100 (20 Tasks) with 500 memory. We report the average and standard deviation over 5 trials

| **Method** | Acc ↑ | $AAA^{val}$ ↑ | WC-Acc$^{val}$ ↑ | Probed Acc ↑ |
|---|---|---|---|---|
| $i.i.d$ | 39.0 ± 1.8 | - | - | 49.3 ± 0.9 |
| GDumb | 39.6 ± 0.4 | - | - | - |
| AGEM | 3.1 ± 0.3 | 10.5 ± 0.5 | 3.1 ± 0.2 | 18.6 ± 0.8 |
| ER | 34.9 ± 1.8 | 39.1 ± 1.7 | 13.2 ± 0.8 | 48.7 ± 0.7 |
| ER + LwF | 36.7 ± 1.3 | 41.7 ± 1.8 | 17.2 ± 0.9 | 48.5 ± 0.9 |
| MIR | 31.8 ± 1.4 | 33.6 ± 2.6 | 8.4 ± 1.4 | 47.8 ± 0.8 |
| ER-ACE | 35.1 ± 1.2 | 40.6 ± 1.5 | 16.8 ± 1.1 | 47.0 ± 0.7 |
| DER++ | 36.1 ± 1.7 | 40.8 ± 2.0 | 14.6 ± 0.4 | 49.1 ± 0.7 |
| RAR | 36.9 ± 2.0 | 42.2 ± 1.3 | 16.1 ± 1.2 | 48.1 ± 1.2 |
| SCR | **43.5** ± 0.7 | 50.2 ± 2.1 | **32.6** ± 0.7 | 47.3 ± 0.7 |
| ER[‡] | 43.0 ± 0.7 | **52.7** ± 2.2 | 30.7 ± 0.9 | **49.5** ± 1.4 |

Table 4.4: Last step results on Split-Cifar100 (20 Tasks) with 8000 memory. We report the average and standard deviation over 5 trials

size as SCR, we see that with this modification, the performance of ER matches the one of SCR, indicating that the performance of SCR in this setting is probably due to the bigger memory batch size and not so much to the supervised-contrastive loss.

[†]Modified ER version with a memory batch size of size 118 (to match the size of the SCR one)

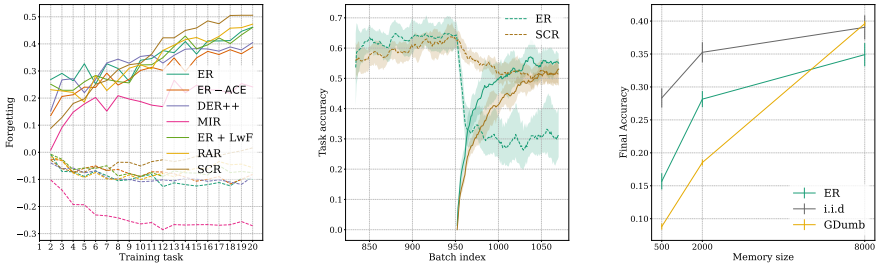


Figure 4.3: **Left:** Forgetting (full lines), and Cumulative Forgetting (dotted lines) on Split-Cifar100 with 2000 memory; **Middle:** Illustration of the difference in stability between ER and SCR on Split-Cifar100 (20 tasks), using 2000 memory. We place ourselves at the task shift between task 4 and 5 and display the accuracy on previous task data (dotted lines) as well as the accuracy on current task data (full lines).; **Right:** Final performance of ER, i.i.d. reference method, and GDumb baseline for 3 different memory sizes on Split-Cifar100

## 4.7 Related Work

**Existing surveys.** Surveys on continual learning have focused on different aspects. Parisi et al. [125] provide a survey on lifelong learning and draw parallels with how biological systems prevent catastrophic forgetting. The survey of Lesort et al. [94] studies continual learning focusing on robotics applications. More recent surveys have focused on popular settings for continual learning. De Lange et al. [39] studies task-incremental learning, and Masana et al. [111] investigates class-incremental learning. More similar to the proposed work in this chapter, is the work of Mai et al. [108] who propose an empirical evaluation of several online continual learning methods. However, other than them, we here aim to compare a variety of competitive replay methods that each use different approaches to tackle the setting of online continual learning (as described in Figure 4.1). On top of that, we evaluate and compare the performance of each method from different points of view, analyzing both the final performance but also the stability (by evaluating on the validation set after each mini-batch). We also evaluate the learned representation by probing the features with the full dataset at the end of the training, as in [38].

**Existing Libraries.** This survey contributes to the implementation of multiple methods in the Avalanche Library [26]. Avalanche is an end-to-end library based on Pytorch

with the goal of providing a codebase for fast prototyping, training, and reproducible evaluation of continual learning algorithms. Besides this, other libraries have been implemented with different objectives and qualities. SequeL [45] is a new library that focuses on developing methods not only in PyTorch, but also in JAX. It provides a simple interface for running experiments in both. However, the newness of the library and the need to implement the methods in both languages make it difficult to use. On the other hand, Sequoia [120] is a library that attempts to unite as many continual learning settings as possible under a common tree. The root is the most difficult problem to learn, and the leaves and branches are different settings. Lastly, Continuum [49] is a library that focuses mostly on the benchmark aspects of continual learning and provides tools to easily split the datasets and iterate on the resulting tasks.

**Additional methods.** In addition to the methods implemented, there are other methods proposed in recent years. In Online Bias Correction [35], the authors explain how experience replay biases the model output towards recent observations. With this, they propose a way to modify the classifier output and mitigate the bias. Following the same idea of reducing the bias, Guo et al. [62] propose OCM based on mutual information maximization. Here, the authors deal with the bias reduction caused by cross entropy and they encourage the preservation of previous knowledge. Another approach that relies on a buffer is Proxy-based Contrastive Replay [97]. Here, the authors propose a way to complement a buffer loss and a contrastive loss. Using a Visual Transformer in conjunction with a focal contrastive learning strategy, Wang et al. [154] suggest mitigating the stability-plasticity dilemma.

## 4.8 Conclusions

In this study, we examined the performance of various Online Continual Learning (OCL) methods, focusing on performance, stability, representation quality, and forgetting. Our analysis revealed intriguing insights. Firstly, we found that stability does not always translate to higher accuracy, challenging the notion that a stable model guarantees superior performance in the OCL setting. Additionally, we observed that the quality of the representation learned by continual learning methods does not differ strongly from the one obtained by learning on the i.i.d. stream, indicating that the main challenge faced by continual learning methods is to learn a good classifier. We also found that methods were prone to underfitting in the OCL setting, challenging the common assumption that continual learning methods suffer from forgetting; we here claim that they keep improving their performance on previous tasks as they learn on subsequent tasks. In general, we found all compared methods to perform very similarly to the common Experience Replay (ER) method. We also investigate

some small implementation differences and conclude that sometimes small details in implementations can make a method shine using the existing metrics, but that it is often possible to obtain these same results by slightly modifying the baseline ER hyperparameters or implementation details, highlighting the necessity to implement these methods in a unified framework like avalanche so that they can be more fairly compared. Finally, we found that no single OCL method proved to be universally superior across all metrics or memory sizes, highlighting the absence of a one-fits-all solution. Considering these findings, we identify several promising research directions for online continual learning.

We advocate for stronger connections between normal i.i.d. online learning and online continual learning, given their similar representation strengths. As the ER strategy is already common to both settings and proven competitive, emphasis should be put on tuning its hyperparameters during training, as already attempted in [168] and [24]. Proper hyperparameter tuning in online continual learning remains an open challenge. Additionally, we encourage further exploration of linking stability metrics to training efficiency, as we found that poor stability does not necessarily impact final representation strength. If no direct link exists, enforcing good stability during training may not be essential, and ad-hoc methods like the one we use in Chapter 3 could be sufficient to achieve desired stability.

# 5 An Empirical Analysis of Forgetting in Pre-trained Models with Incremental Low-Rank Updates*

## 5.1 Introduction

Over the years, the increase in scale of deep models has been instrumental in driving progress. It has now become inevitable for practitioners to embrace huge pretrained models (such as Llama [151] or CLIP [132]) in the pursuit of efficient learning solutions. However, resuming training on these large models can be very demanding in terms of computational resources, and often necessitates the use of techniques that alleviate the computational and memory burden when training.

Low-rank training techniques [3, 70, 96] can effectively finetune enormous pretrained models (up to 175 billion parameters for GPT 3.5 [20]), by only tuning a few parameters (up to 10,000 times less) to later project them into the full parameter space (or only a subset of parameter space). This allows practitioners to train models using little memory. The most popular technique is Low-Rank Adaptation (LoRA) and is extensively used to finetune large pretrained models because of its simplicity and ease of use [70]. LoRA learns a low-rank version of the weight matrices of a given network and sums them to the existing network parameters. The adapters learned in this way can then be "plugged" into and "unplugged" from the pretrained model by simply adding or subtracting the adapter weights to or from the pretrained ones. Moreover, several adapters can be combined together [33, 74] by summing the adapter weights together. Parameter-efficient learning methods like LoRA were initially studied and successfully applied in the context of transfer learning, where the task is to optimize for a single target domain.

While pretrained models demonstrate remarkable capabilities and can be adjusted to downstream tasks in a one-step fashion, current transfer learning methods based on LoRA have not investigated the loss (catastrophic forgetting) of the capabilities of the pretrained model during the LoRA adaptation. In addition, transfer learning only considers adaptation to a single target domain, whereas in many realistic applications models should continually adapt to changing data distributions, varying domains, temporal changes, a larger set of semantic classes, etc. The field of continual learning

*This chapter is based on an article that is under review

investigates methods to address these situations, where the aim is to adapt to a changing non-IID distribution while accumulating the knowledge of all domains. However, the vast majority of continual learning literature has considered training from scratch [39, 82, 96, 111]. Recently, starting from pretrained models is gaining some attention in the community [60, 124, 155, 162]. Unfortunately, most existing works do not aim to improve the performance of the pretrained model and focus on optimizing the performance on the sequence of downstream tasks. Only a few recent works consider continual improvement of the pretrained model [36].

We think that with the growing importance of very large pretrained models (or foundation models) there will be a further shift towards studying parameter-efficient continual learning algorithms. Therefore, we propose to study the consequences in terms of forgetting and knowledge transfer of incremental LoRA updates. To study incremental updates of LoRA to a sequence of tasks, we propose an experimental setup in which we incrementally learn four fine-grained tasks (Cars, Flowers, Aircraft, and Birds) starting from a model pretrained on ImageNet. Contrary to many previous works, we do not aim only to achieve good performance on downstream tasks, but we also aim to maintain the capacity on the pretrained foundation task (i.e. the 1,000 ImageNet classes used for pretraining in our experiments).

Our contributions are the following:

- We explore the interplay between low-rank updates of pretrained model and continual learning, and we show that lower rank updates leads to less forgetting.
- We observe that the forgetting obtained for vision transformers (ViT) can be interpreted as a form of contextual forgetting, where the forgotten samples correspond to ImageNet classes related to the domain of the fine-grained datasets.
- We evaluate incremental LoRA updates on both Vision Transformer (ViT) and Residual Network (ResNet) models and show that convolutional networks suffer far more forgetting than Vision Transformers.

## 5.2 Related Work

Incremental learning refers to the paradigm of updating an existing model with a continuous stream of data or tasks, while mitigating the phenomenon of catastrophic forgetting. Traditional incremental learning methods initialize the feature extractor in one of two ways: the *Cold-Start* scenario, where the feature extractor is randomly initialized before the incremental learning steps [82,96,98,107,140], or the *Warm-Start* scenario, where the feature extractor is pre-trained on half of the available dataset before being incrementally trained on subsequent tasks [60, 129, 172–174].

However, the exceptional performance of large foundational models, such as Vision Transformers (ViT) [48] for vision tasks, and BERT models [42, 102, 142] for natural language processing tasks, along with self-supervised techniques [13] for effectively pre-training these models, has sparked an increasing interest in exploring incremental learning with *Pre-Trained* architectures. Focusing on vision tasks, recent state-of-the-art methods leverage the powerful representational capabilities of ResNet architecture [68] and ViT models, both pre-trained on the Imagenet dataset [41], to further enhance incremental learning outcomes.

Panos et al. [124] adapt a pre-trained ResNet for the first task, then freeze the feature extractor and classify the classes across incremental steps using Linear Discriminant Analysis (LDA). Wu et al. [162] propose to dynamically expand a frozen pre-trained ResNet model with new convolutional layers, fine-tuning it on new data and fuse the previous and current classifier for each new class using exemplars. Wang et al. [155] propose learning to dynamically prompt a pre-trained ViT to sequentially learn tasks. Goswami et al. [60] freeze a pre-trained ViT feature extractor after the first task and then use the class covariance matrix and the class means obtained by the feature extractor to perform classification on all task classes. Zhang et al. [166] fine-tune the ViT feature extractor during training and, at the end of each task, calibrate the task classifiers for class-incremental evaluation using Gaussian Prototypes accumulated across all encountered tasks.

Some recent works explore the usage of low-rank training for updating a pre-trained feature extractor in incremental learning. Hyder et al. [73] perform *Continual Learning via Low-Rank updates*, where they represent the full weight matrix as a sum of rank 1 matrices each corresponding to one task. When they learn a new task, they learn a new rank 1 matrix along with a selector vector that weights the existing matrices to better fit the current task, this method allows them to have zero forgetting in the task aware setting. Witsuba et al. [159] start from a pretrained model and incrementally learns per task LoRA adapters, they further select the best LoRA adapter to be applied at inference by performing task-inference via task-wise prototypes storage and retrieval. Chitale et al. [33] propose to train a LoRA per task, then at the end of each task merge the learning low-rank parameters with the frozen feature extractor and finally tune the overall model using exemplars for class-incremental evaluation.

Given the recent impressive efficacy of pre-trained models in incremental learning, current research is directed towards assessing how pretrained initialization influences the phenomenon of forgetting during the incremental learning process. These studies seek to understand the relationship between the representations learned for the pre-

trained (upstream) classes and those for the subsequently added (downstream) classes, as well as the extent to which pre-training methods impact overall performance. Ramasesh et al. [133] empirically evaluate the catastrophic forgetting on ResNet and Transformers architecture. They show that pretrained networks are less susceptible of forgetting and that their robustness improves with scale of both model and pre-training dataset size. Mehta et al. [112] empirically demonstrate that pretrained initialization helps mitigate forgetting, and this effect is associated with wider minima in the loss landscape.

Lee et al. [92] examine the extent to which the architecture of pretrained models and the pre-training method, whether supervised or self-supervised, influence incremental learning algorithm performance. Their analysis reveals that an under-performing incremental learning algorithm, originally designed for traditional incremental scenarios (Warm-Start or Cold-Start), can be enhanced to compete with, or even surpass, existing state-of-the-art solutions when evaluated with pretrained initialisation. Moreover, they find that combining minimal regularization with exemplars is more beneficial than stronger regularization when starting with a pretrained network. Janson et al. [75] shows that using a Nearest-Mean-Classifier rule on top of a frozen pretrained ViT provides competitive results with recent state-of-the-art approaches.

Galashov et al. [54] discuss the impact of pretrained foundation models in both supervised and self-supervised settings, demonstrating simply fine-tuning the classification heads on a top of a foundation model in task-incremental setting the final performance surpass those of models fully trained from scratch. Ostapenko et al. [121] examine the efficacy of latent replay, a technique involving the replay of a network's intermediate representations across tasks. Their findings indicate that latent replay surpasses the efficiency of experience replay, which relies on replaying exemplars. The effectiveness of latent replay, however, hinges on the correlation between downstream and upstream classes. Specifically, latent replay exhibits reduced transfer effectiveness in out-of-distribution incremental scenarios, such as when transitioning from an ImageNet upstream task to a Cars downstream task. Conversely, it demonstrates notable transfer benefits in in-distribution domains, exemplified by the transition from an ImageNet upstream task to a SplitCifar100 downstream task.

## 5.3 Low-Rank Training

Low-Rank training for neural networks is based on the idea that while the parameter space of a network is so big, it is possible to find another parametrization of the network so that we learn weights of lower dimension and later project it back into the full parameter space. Li et al. [95] found that using this technique, it is possible

to train overparametrized network from scratch by sometimes only learning a few thousands of parameters that are then projected back to the full parameter dimension. Aghajanyan et al. [3] later show that if we consider a pretrained model, the dimension of the additional parameters that is required to solve a downstream task gets lower the better the pretrain model is. Hu et al. [70] exploit this property and propose to learn low-rank weight matrices instead of a flat parameter vector projected back to the full parameter space. Using this technique called LoRA for Low-Rank Adaptation, they can find solutions to downstream tasks when starting from a pretrained model by learning matrices of rank as low as 1, drastically reducing the number of parameters to be learnt.

LoRA [70] is a low-rank training method that has been designed initially for large language models (LLMs). The principle of this method is quite straightforward, in order to ensure that the difference between the finetuned model and the pretrain model is low-rank, they parametrize this difference $\Delta W_{finetune}$ as the product of two matrices $B^{n\times r}$ and $A^{r\times m}$ with a small $r << min(n, m)$, this product has a rank less than or equal to $r$ by construction. While LoRA has been initially designed to work for 2 dimensional parameter tensors (matrices), which is the case for query key value parameter matrix of the transformers, it can be also applied on convolution layers of arbitrary kernel size with a small adaptation. However, since in this chapter we apply it on convolution layers of kernel size 1x1 (third convolution layer of each ResNet block), we can apply LoRA just the same way than on query key value matrices by viewing the $1 \times 1 \times n \times m$ kernel as an $n \times m$ matrix.

$$\begin{aligned} W &= W^{n\times m}_{pretrain} + \Delta W^{n\times m}_{finetune} \\ &= W^{n\times m}_{pretrain} + B^{n\times r} A^{r\times m} \end{aligned}$$

Existing works that apply LoRA in continual learning [159] use one LoRA adapter per task and rehearse it at inference, [33] merge the learned LoRA and later finetune the model using stored exemplars. Approaches that store the LoRA for later rehearsal make the choice to sacrifice transfer between tasks in order to have zero forgetting. Here we want to study the forgetting induced by continually learning the pretrained model, we hence choose a natural approach which is to learn one LoRA adapter per task that we sum to the model weights at the end of each task, without storing it separately. We believe this option makes sense for the tasks we consider that are fine-grained, however we could imagine more complex schemes learning different LoRAs for different samples (for instance one LoRA per class), in order to have one LoRA of low-rank specialized to each domain, we leave this to future works.

## 5.4 Experiments

### 5.4.1 Experimental Setup

**Datasets:** We perform the continual learning experiments by learning on a sequence of 4 fine-grained classification datasets with varied topics and dataset size. Stanford Cars [84] is comprised of 196 categories of cars, and contain 8,144 training images and 8,041 test images. Oxford 102 Flowers dataset [119] has 102 categories of flowers with around 1,000 training images. FGVC-Aircraft dataset [110] is comprised of a total of 10,200 images of 102 aircraft model variants. Caltech-UCSD birds dataset [156] contains 6033 images of 200 birds species. For all the datasets, we apply the same transformations during training and evaluation than the ones that were applied to Imagenet during training of the specific model. We consider two training settings using these datasets, one of them we coin the *Short Setting* considers training on the sequence of 4 tasks Cars - Flowers - Aircraft - Birds, while the other setting we coin the *Long Setting* considers training on a two repetitions of this sequence, and splits each dataset in two random parts, to create a sequence of 8 training experiences by revisiting the previous tasks.

**Models:** To get the pretrained models, we use the timm [158] library and fetch pretrained model from the HuggingFace database. In particular, and in order to provide a more complete analysis, we use two models, one ViT [47] with patch size 16 (HuggingFace model id: "timm/vit_base_patch16_224.augreg_in1k") , with a total of 86M parameters, and one ResNet-50 [68] (HuggingFace model id: "timm/resnet50.a1_in1k"), with 21M parameters, both pretrained on Imagenet 1k [41]. Both model have similar performance on the Imagenet test set, with the ViT having 79.14% accuracy and the ResNet-50 80.04% Top-1 accuracy.

**Low-Rank Training:** In order to perform low-rank incremental training, we setup a new LoRA adapter for each task that we merge into the model weights at the end of training each task by simply summing the adapter weights to the model weights. For the ViT, we choose to apply the adapter on every attention matrix (query, key, value weight matrix), which is a classical process. For the ResNet, it is less obvious how to choose the parameters to adapt since adapting every convolutional layer would results in prohibitive amount of adapted parameters. In order to provide a fair comparison with the ViT, we choose to adapt only the last convolutional layer of each block ($1 \times 1$ convolution), which results in an amount of modifiable parameters that roughly matches the one of the ViT attention matrices (1/4th of the total number of parameters of the network). We then vary the rank of the LoRA which in turns varies the number of

trainable parameters, we provide in 5.1 a comparison on how this number of trainable parameters evolve linearly but with different slopes for the chosen ResNet and ViT. We insist here that although the number of trainable parameter varies with the rank, the number of modifiable parameters does not depend on the rank and is always fixed once we have chosen the type of layer to modify. Additionally, we tune the parameter alpha of the LoRA to be equal to two times the LoRA rank, we make this choice in order to be able to work at fixed learning rate while fairly comparing results across ranks, since fixing alpha and increasing the rank leads to smaller learning rate for bigger ranks, which also has an effect on learning and forgetting. We choose a range of rank of [1, 2, 3, 4, 5, 6, 7] for ResNet-50 and [1, 2, 4, 6, 8, 16, 32] for ViT. We decided to expand the range more for ViT based on the observation that forgetting was lower for lower ranks of LoRA in combination with ViT, whereas forgetting was quite high for ResNet-50 even when using low-rank, and increasing the rank makes it even worse as seen in Figure 5.2.

**Optimization:** We use the AdamW optimizer [81] with learning rate of $1e-3$, a batch size of 32 and a cosine learning rate scheduler with warmup that we restart for each task, we choose the warmup steps to be always equal to 1/18 th of the total number of finetuning steps for one task, and we roughly tune the number of epochs for each task so that we train for a bit more epochs on tasks that are very small (in order to get a decent number of training iterations and converging training accuracy).

**Reproductibility:** To perform our continual learning experiments, we use the avalanche library [26, 103], which is an open source library available at https://github.com/ContinualAI/avalanche. We will release the code to reproduce our experiments upon acceptance.

### 5.4.2 Results

**Impact of forgetting the pretrain task:** To assess the impact of forgetting of the pretrain task on further transfer capacity, we compare the results of finetuning the pretrained model directly on each fine-grained task (i.e. *Direct Transfer*) versus finetuning it continually, by taking the checkpoint at the end of learning each task to start training on the subsequent task. In order to fairly compare it to later experiments done with LoRA, we finetune only the query key value matrices for ViT and the third convolutional layer of each block for ResNets. We present the results of this experiment in Table 5.1. We see that after learning 'Cars' and 'Flowers' the network only obtains a 30% performance on 'AirCraft', whereas a direct adaptation to 'Aircarft' would yield 58%, showing that much of the pretrained knowledge useful for airplane

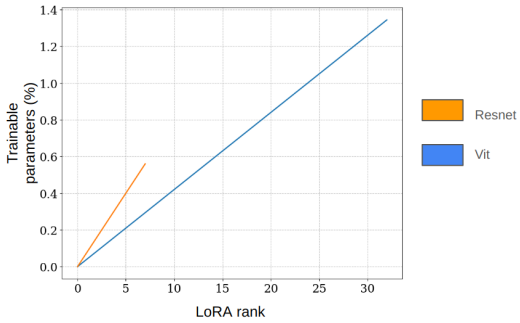


Figure 5.1: Comparison of the number of trainable parameters as a function of the LoRA rank for both ResNet-50 and ViT (in % of the total parameters). We restrict the comparison to the range of considered ranks.

classfiication has been lost by the two finetuning stages. In order to avoid that while still continually learning the pretrained model, we need to avoid forgetting of the pretrain task.

| Task | Continual FT | Direct Transfer |
|---|---|---|
| Cars | 83.0 | 83.0 |
| Flowers | 86.1 (-**4.2**) | 90.3 |
| AirCraft | 30.0 (-**28.6**) | 58.6 |
| Birds | 36.8 (-**36.9**) | 73.7 |

Table 5.1: Comparison between the (first) accuracy obtained on downstream tasks when performing continual finetuning on the task sequence Cars - Flowers - AirCraft - Birds versus when performing direct transfer learning on each dataset, starting from the pretrained ViT model.

**Does the LoRA rank impact forgetting?** We experiment with various ranks of the LoRA adapter that we train and merge for each task. We then report the accuracy and forgetting on the pretrained task (Imagenet 1k [41]) as well as on the downstream tasks. Additionally, we apply LwF [96], a continual learning method that could realistically be applied without access to the pretraining task data, on top of the low-rank learning procedure, in order to confirm that the gains obtained by low-rank learning are orthogonal to the gains obtained using existing continual learning method.

We experiment with ranks in the range [1, 2, 4, 6, 8, 16, 32] for ViT and [1, 2, 3, 4, 5, 6, 7] for ResNet (See Discussion in Section 5.4.1).

We present the results for both continual low-rank finetuning and LwF in Figure 5.2 and Figure 5.3 respectively, and the detailed results for finetuning in Table 5.2. In Figure 5.2 we see that the rank chosen for the adapter has a significant impact on the forgetting of both the pretraining task and on the downstream task Cars (this conclusion generally holds for all tasks in the sequence), this impact also seems more important for ViT than for ResNet. We see that while learning an adapter of higher rank, the accuracy on the downstream task is positively affected (at least up to the maximum rank considered here), but choosing a very low-rank already reaches very satisfying accuracy compared to training only the output head, providing a good stability-plasticity tradeoff. In Figure 5.3, we see that when applying LwF in combination with LoRA, we still see a big discrepancy between LoRA adapters of different ranks, although the forgetting is still drastically reduced for all of the ranks. These results indicate that the rank has an impact on forgetting that is orthogonal to the one of LwF. Interestingly, by combining ViT with an adapter of rank 1 and LwF, we even witness some slight backward transfer on Imagenet when learning on Cars with the accuracy increasing by a small amount of 0.1%. We also see in Table 5.2 that the gains in plasticity obtained by training adapter of higher rank do not manage to convert in higher overall average accuracy since it is counterbalanced by too high forgetting, it results in a final average accuracy that diminishes with the rank.

| Rank | Average Accuracy | Average Forgetting |
|---|---|---|
| 1 | 69.9 | 9.2 |
| 4 | 66.8 | 14.3 |
| 8 | 68.0 | 13.6 |
| 16 | 62.2 | 21.0 |
| 32 | 55.9 | 28.9 |

| Rank | Average Accuracy | Average Forgetting |
|---|---|---|
| 1 | 52.2 | 23.6 |
| 3 | 51.2 | 25.5 |
| 4 | 52.0 | 24.8 |
| 5 | 50.3 | 27.8 |
| 7 | 47.9 | 30.7 |

Table 5.2: Final average accuracy and forgetting after learning on the *Short Setting*, depending on the rank, for ViT (Left) and ResNet (Right)

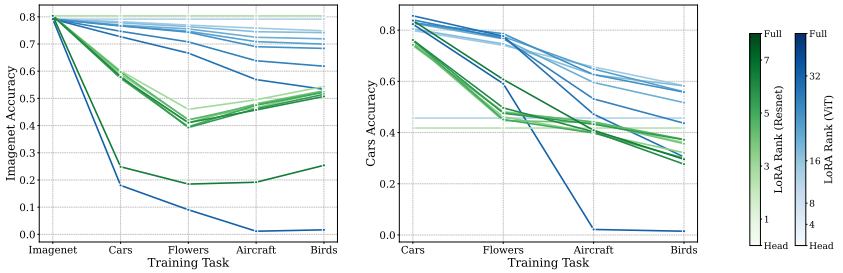


Figure 5.2: Test Accuracy on the pretrain task Imagenet1k (Left) and on the downstream task Cars (Right) when fine tuning on the *Short Setting* with multiple LoRA ranks, using ViT network (Blue) and ResNet-50 (Green).

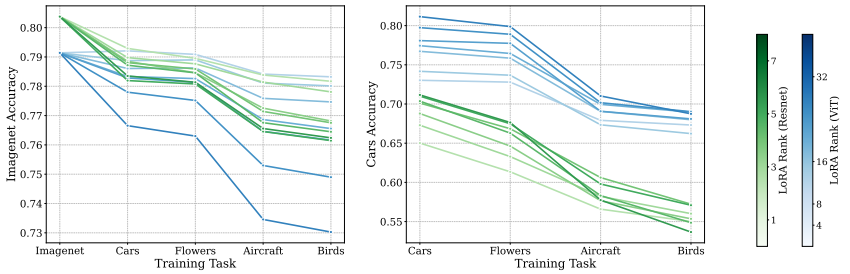


Figure 5.3: Test Accuracy on the pretrain task Imagenet1k (Left) and on the downstream task Cars (Right) when training on the *Short Setting* with multiple LoRA ranks in combination with the LwF method, using ViT network (Blue) and ResNet-50 (Green).

**Interpretation of the forgetting of the pretraining task:** Since the tasks that we choose have a very specific domain and the pretrain task covers a wide variety of domains with some categories that lie in the domain of the fine-grained tasks, we try to interpret the forgetting of the pretraining task by looking at the most forgotten categories of Imagenet1k. Whenever learning on a new downstream task, we report the new errors that the model make on Imagenet. In this experiment, we exclusively consider *direct transfer* from the pretrained model using a LoRA adapter, to maintain clarity and avoid confounding variables. We present these differences in error vector by using two methods, the first method simply considers the histogram of the most affected categories, while the second one considers comparing the distributions of

the semantic distances between the category name and a target word that describes the domain of the fine-grained dataset. We chose to report the Wu-Palmer semantic distance which is widely used in the field of Natural Language Processing [164] .

In Figure 5.4, we show the most forgotten Imagenet categories after learning on the Cars dataset, using ViT with a LoRA adapter of rank 32, with a total forgetting of 7% on Imagenet test set. Looking at the category names, we see that the most forgotten one is "sports cars" from Imagenet, which indeed is very related to the Cars dataset domain. While not all categories in the histogram are related with cars, there is a good amount of them that are, among them "grille", "tow trucks", "convertible", "minivan", "limousine", "pickup" are all present in the top 20 forgotten categories, which provides strong evidence that there is contextual forgetting. The same conclusion holds for Aircrafts, where the first forgotten Imagenet category is "airliner". In Figure 5.5, we see that while Imagenet categories are generally lying in two "modes" of similarity with the word plane and cars, the newly forgotten categories mainly lie in the mode that is most similar to the target word, with a small spike on words that are very similar, which corresponds to the most forgotten categories that we observe in Figure 5.4. Interestingly, we do not observe such contextual forgetting for ResNet-50 model (See Figure 5.6).

We hypothesise that this contextual forgetting happens because of a bigger feature drift for images which category is related to the downstream task at hand, which results in bigger forgetting due to the missing head adaptation, although the network might now be better at distinguishing these categories since it "improved" this part of the feature space.

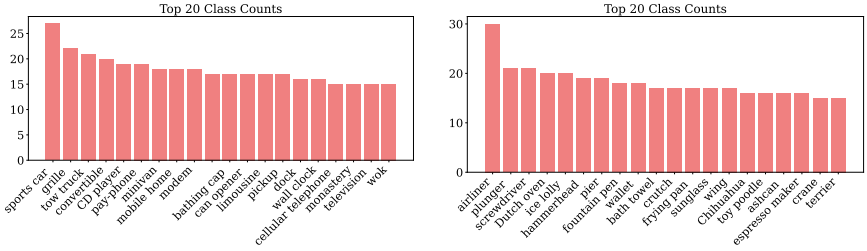


Figure 5.4: 20 most forgotten categories of Imagenet1k after learning on Stanford Cars (Left) and Aircraft (Right), using direct transfer from the pretrained ViT model with a LoRA adapter of rank 32

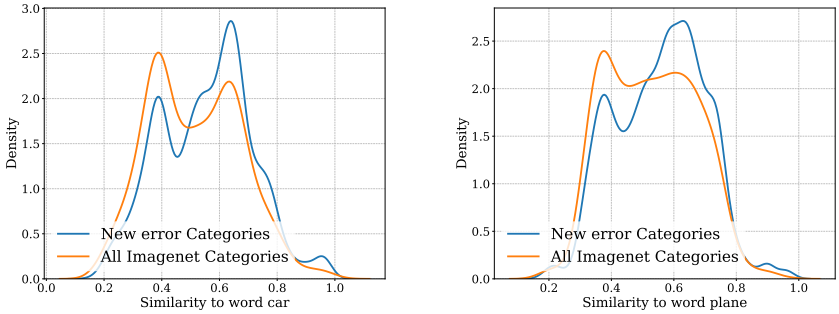


Figure 5.5: Wu-Palmer similarity between the forgotten category names an the word "Car" after learning on Cars dataset (Left) and with the word "plane" after learning on Aircraft dataset (Right). Comparison of the similarity distribution of all Imagenet categories versus the forgotten categories (weighted by the amount of test images forgotten in this category), when using direct transfer from the pretrained ViT model with a LoRA adapter of rank 32

Although we observed contextual forgetting quite clearly for the ViT model, we did not find evidence of it in the ResNet-50 model. In Figure 5.6, we perform the same analysis as we did in Figure 5.4 and 5.5 but for ResNet-50 instead of ViT. In order to have the most chances to observe this contextual forgetting, we choose the finetuning run that had the lowest forgetting using a LoRA of rank 1, which results in 60% accuracy on Imagenet after learning on Cars. However, even when doing so, we find no evidence of contextual forgetting for this network. Indeed, for that network, the most forgotten categories do not seem very related to the downstream task that has been trained, and the distribution of new errors almost matches the distribution of all Imagenet categories which shows no bias towards forgetting of classes semantically related to the downstream task domain.

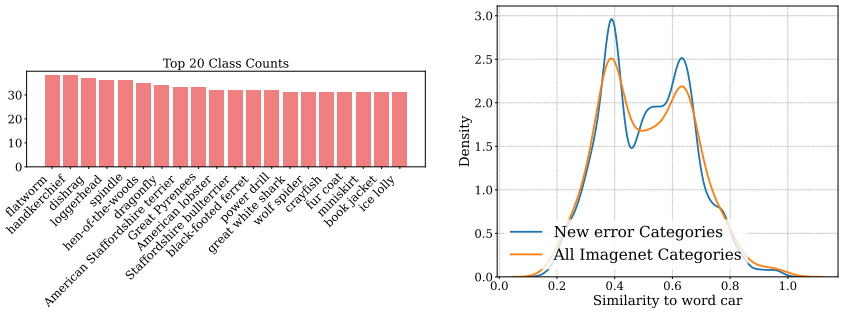


Figure 5.6: (Left) 20 most forgotten classes of Imagenet1k after learning on Stanford Cars, (Right) Wu-Palmer similarity between the forgotten category names an the word "Car". Comparison of the similarity distribution of all Imagenet categories versus the forgotten categories (weighted by the amount of test images forgotten in this category), when using direct transfer from the pretrained ResNet-50 model with a LoRA adapter of rank 1

**Can we improve the forward transfer capabilities of the base model?:** In order to answer this question, we use the *Long Setting* described in the Dataset section, which revisits each fine-grained task one more time, but can only learn with half of the data every time. In order to see if forward transfer is possible we are interested in looking at the accuracy reached the second time the task is encountered and compare it to the accuracy reached when the task is encountered the first time. If we can reach a better accuracy when encountering the second part of the dataset, it means that forward transfer is effective.

In Figure 5.7, we show the results on the *Long Setting*. Curiously in that case, we get more occurrences of backward transfer on Imagenet1k, where the accuracy initially goes down but goes back up at some point in the stream. We also notice that there is knowledge transfer between both half of the downstream task dataset both for ViTs and ResNet. For instance for the Aircraft task, the first accuracy sets up at around 45% for ViT and 30% for ResNet, and jumps to around 52% for ViT and 45% for ResNet when the task is encountered for the second time. This hints at the fact that we actually managed to improve the forward transfer capabilities of the model by training on the first half of the dataset so that we can reach better accuracy on the second half, which is an encouraging step regarding the possibility to continually improve these models. We also see in Figure 5.8 that the final average accuracy over all tasks is not affected by the *Long Setting* for ViTs trained with very low-rank adapters, and even improved

for ViTs trained with adapters of higher rank. In comparison, the accuracy for ResNet is overall negatively affected (around 3% drop in average accuracy between the two settings).

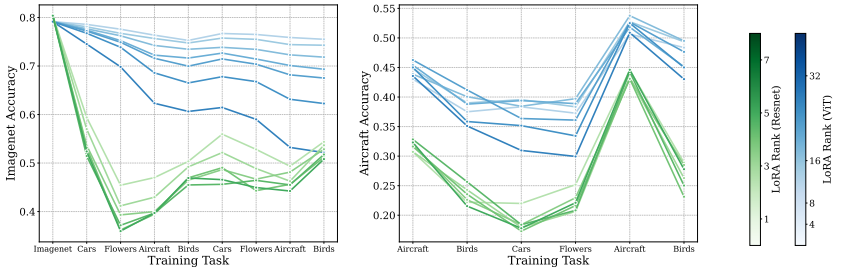


Figure 5.7: Test Accuracy on the pretrain task Imagenet1k (Left) and on the downstream task Aircraft (Right) when fine tuning on the *Long Setting* with multiple LoRA ranks, using ViT network (Blue) and ResNet-50 (Green).

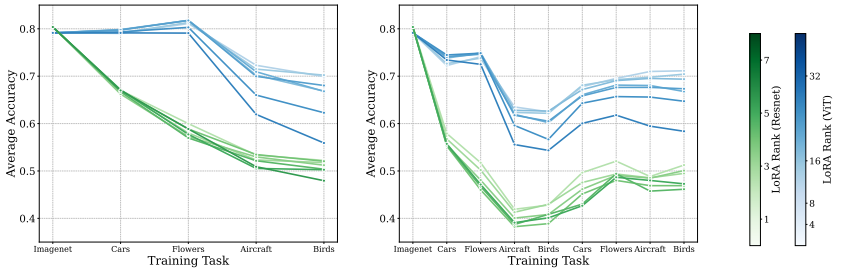


Figure 5.8: Average test accuracy across all seen tasks on the *Short Setting* (Left) and on the *Long Setting* (Right) when fine tuning with multiple LoRA ranks, using ViT network (Blue) and ResNet-50 (Green)

## 5.5 Conclusion

In this chapter we explored the possibility of continually training large pretrained models by finetuning them on small, fine-grained downstream tasks using low-rank adapters that we merge at the end of each task. While doing so, we monitor the performance of the pretraining task in order to determine how much the learning of downstream tasks affects it. We make a few of interesting observations.

Firstly, we see that varying the rank of the low-rank adapter has an important impact on forgetting of both the pretrain task and the downstream tasks, and that in general forgetting diminishes when the rank of the adapter is low. These gains in stability are still present when combined with the existing continual learning method LwF, which is a method that can be applied without access to the pretrain task dataset.

Secondly, we observe that when finetuned on fine-grained downstream tasks with a specific domain, ViTs exhibit *contextual forgetting* in which the pretrain task categories that are forgotten are semantically related to the downstream tasks on which it has been finetuned.

Lastly, we observed many differences in the application of LoRA to ViTs and ResNets. In particular, we do not observe *contextual forgetting* in ResNets, but we do observe a drop in performance for ResNet training in the *Long Setting* whereas ViT conserves similar accuracies to the *Short Setting*. In general, we also find ResNet to suffer more from forgetting in the context of low-rank learning (while it suffers less in case of full finetuning). This suggests that transformer-like architectures are more adept at accumulating knowledge through incremental low-rank updates.

Overall, we believe that using per-task low-rank updates in order to incrementally improve pretrained models is a promising direction and we hope that future research can exploit our analysis to further explore it.

# 6 Conclusions and Future Work

## 6.1 Conclusions

In this thesis, we analyzed various continual learning settings ranging from offline learning from scratch, to online learning from scratch, to offline learning from a big pretrained model. Every time, we aimed to understand the reasons that lead to a performance gap when comparing continual learning to learning from scratch on the concatenation of the training data coming from the continual stream.

**In Chapter 2,** we made it clear how the difficulty to learn cross-task features in the class-incremental setting contributes to the performance gap. We also put this in relation to the commonly used *forgetting* metric, and show that this metric does not correctly capture the challenges of class-incremental learning. To remediate to that, we propose a new metric coined "cumulative forgetting" that takes into account the increasing difficulty of the class-incremental task.

**In Chapter 3,** we investigate the stability gap and question its potential impact on the optimization process. Using a temporal ensembling method, we show considerable progress on metrics that were designed to measure the stability gap, without influencing the training process. We also show that in the case of online learning, the use of temporal ensembling can drastically improve the final performance and address the *task-recency bias*.

**In Chapter 4,** we take a deeper look into existing online continual learning methods and compare them in a fair way using the widely used continual learning open source library Avalanche. Using an extensive array of metrics including stability and representation probing metrics and comparing the continual learning methods to naive training done on the i.i.d stream, we uncover that they both lead to representations of similar strength. We see that the main problem faced by online methods is the one of continually adapting the classifier, and encourage future works to build more links between *online learning* and *continual online learning*.

**In Chapter 5,** we investigate a different problem which is continual learning of strong pretrained, or "foundation" models. In that context, we are interested in

evaluating the impact of the rank when updating the model using low-rank updates, since this is a common technique used to fine-tune large pretrained models. We discover that the rank used for training indeed has a strong impact on forgetting, with lower rank inducing lower forgetting. Through this analysis, we also observe a curious behaviour in Vision Transformer models that we name "contextual forgetting", where the forgetting caused by fine-tuning on fine-grained datasets can be interpreted in terms of new errors on the pretrain task, since we observe that new errors belong to categories that are semantically related to the domain of the fine-tuning task.

## 6.2 Future Directions

The observations from Chapter 2 that the learning of cross-task features is an important cause of the performance gap encourages future research to focus on the learning of more general features that would not suffer from that problem. Although learning unsupervised features might initially lead to a performance drop because these features are not specific enough for the discriminative task, they might be better suited for continual learning and outperform supervised learning approach when put to the appropriate scale by using different benchmarks with more training data in each task.

The observations from Chapter 3 make us realize that the stability gap can be artificially solved by using some ensembling techniques at evaluation time. Knowing that, it would be very interesting to discover whether the stability gap has a real lasting impact on the training trajectory and the resulting final performance, in that case it would be necessary for further methods to fill this gap also at training time.

The observations from Chapter 4 encourage stronger ties between online continual learning and classical online learning, since we notice that the representations obtained on the i.i.d stream are of similar strength than the ones obtained on the continual stream, and under-fitting seems to be the bottleneck problem in that setting.

In Chapter 5, we see that low-rank training methods are not only a good tool for classical deep learning but could also have benefits in continual learning, hence they could be used as a component of newly developed methods that aim to continually learn pretrained models. We also make the interesting observation that large pretrained visual transformers exhibit contextual forgetting, this again could be exploited to design methods that try to counter specific instances of forgetting, for instance, we could think of a contextual replay method that only replays instances that correspond to a given context instead of instances from all previous tasks.

# Publications

1. **Soutif–Cormerais, A.**, Masana, M., van de Weijer, J., Bartłomiej, T. (2021). On the importance of cross-task features for class-incremental learning. (The Thirty-eighth International Conference on Machine Learning (ICML) Workshop 2021)

2. **Soutif–Cormerais, A.**, Carta, A., v de Weijer, J. (2022). Improving Online Continual Learning Performance and Stability with Temporal Ensembles. (The Second Conference on Lifelong Learning Agents (CoLLAs) 2023)

3. **Soutif–Cormerais, A.**, Carta, A., Cossu, A., Hurtado, J., Hemati, H., Lomonaco, V., Van de Weijer, J. (2023). A Comprehensive Empirical Evaluation on Online Continual Learning. (International Conference on Computer Vision (ICCV) Workshop 2023)

4. Magistri, S., Trinci, T., **Soutif–Cormerais, A.**, van de Weijer, J., Bagdanov, A.D. (2024). Elastic Feature Consolidation for Cold Start Exemplar-free Incremental Learning. (International Conference on Learning Representations (ICLR) 2024)

5. Goswami, D., **Soutif–Cormerais, A.**, Liu, Y., Kamath, S., Bartłomiej, T., van de Weijer, J. (2024). Resurrecting Old Classes with New Data for Exemplar-Free Continual Learning (The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2024)

6. **Soutif–Cormerais, A.**, Magistri, S., van de Weijer, J., Bagdanov, A.D. (2024). An Empirical Analysis of Forgetting in Pre-trained Models with Incremental Low-Rank Updates (Under review, 2024)

# Bibliography


[1] Stanford, "tiny imagenet challenge, cs231n course.". https://tiny-imagenet.herokuapp.com/, 2015.

[2] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep neural networks. *arXiv preprint arXiv:1711.08856*, 2017.

[3] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

[4] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2017.

[5] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[6] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems (NIPS)*, pages 11849–11860, 2019.

[7] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3366–3375, 2017.

[8] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[9] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 11816–11825, 2019.

[10] Mohammad Abu Alsheikh, Shaowei Lin, Dusit Niyato, and Hwee-Pink Tan. Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Communications Surveys & Tutorials*, 16(4):1996–2018, 2014.

[11] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[12] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems (NIPS)*, 33:3884–3894, 2020.

[13] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, et al. A cookbook of self-supervised learning. *arXiv preprint arXiv:2304.12210*, 2023.

[14] Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000.

[15] Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[16] Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *IEEE International Conference on Computer Vision (ICCV)*, pages 583–592, 2019.

[17] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. A comprehensive study of class incremental learning algorithms for visual tasks. *Neural Networks*, 135:38–54, 2021.

[18] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems (NIPS)*, 24, 2011.

[19] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda

Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems (NIPS)*, 33:1877–1901, 2020.

[21] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

[22] Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. New insights on reducing abrupt representation change in online continual learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[23] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexande Lacoste, David Vazquez, et al. Online fast adaptation and knowledge accumulation: a new approach to continual learning. In *Annual Conference on Neural Information Processing Systems (NIPS)*, 2020.

[24] Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8281–8290, 2021.

[25] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[26] Antonio Carta, Lorenzo Pellegrini, Andrea Cossu, Hamed Hemati, and Vincenzo Lomonaco. Avalanche: A pytorch library for deep continual learning. *arXiv preprint arXiv:2302.01766*, 2023.

[27] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *European Conference on Computer Vision (ECCV)*, pages 233–248, 2018.

[28] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[29] Arslan Chaudhry, Albert Gordo, Puneet Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. In

*Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6993–7001, 2021.

[30] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *International Conference on Learning representations (ICLR)*, 2019.

[31] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, P Dokania, P Torr, and M Ranzato. Continual learning with tiny episodic memories. In *Workshop on Multi-Task and Lifelong Reinforcement Learning*, 2019.

[32] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[33] Rajas Chitale, Ankit Vaidya, Aditya Kane, and Archana Santosh Ghotkar. Task arithmetic with loRA for continual learning. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023)*, 2023.

[34] Aristotelis Chrysakis and Marie-Francine Moens. Online continual learning from imbalanced data. In *International Conference on Machine Learning (ICML)*, pages 1952–1961. PMLR, 2020.

[35] Aristotelis Chrysakis and Marie-Francine Moens. Online bias correction for task-free continual learning. In *The Eleventh International Conference on Learning Representations*, 2022.

[36] Andrea Cossu, Tinne Tuytelaars, Antonio Carta, Lucia Passaro, Vincenzo Lomonaco, and Davide Bacciu. Continual Pre-Training Mitigates Forgetting in Language and Vision. 2022.

[37] Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations (ICLR)*, 2020.

[38] MohammadReza Davari, Nader Asadi, Sudhir Mudur, Rahaf Aljundi, and Eugene Belilovsky. Probing representation forgetting in supervised and unsupervised continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16712–16721, 2022.

[39] Matthias De Lange and Tinne Tuytelaars. Continual prototype evolution: Learning online from non-stationary data streams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8250–8259, 2021.

[40] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *Transactions of Pattern Recognition and Machine Analyses (PAMI)*, 2021.

[41] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

[43] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5138–5146, 2019.

[44] Thomas G Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems: First International Workshop, MCS 2000 Cagliari, Italy, June 21–23, 2000 Proceedings 1*, pages 1–15. Springer, 2000.

[45] Nikolaos Dimitriadis, Francois Fleuret, and Pascal Frossard. Sequel: A continual learning library in pytorch and jax. *arXiv preprint arXiv:2304.10857*, 2023.

[46] Thang Doan, Seyed Iman Mirzadeh, and Mehrdad Farajtabar. Continual learning beyond a single model, 2022.

[47] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2020.

[48] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[49] Arthur Douillard and Timothée Lesort. Continuum: Simple management of complex continual learning scenarios, 2021.

[50] Enrico Fini, Victor G Turrisi Da Costa, Xavier Alameda-Pineda, Elisa Ricci, Karteek Alahari, and Julien Mairal. Self-supervised models are continual learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9621–9630, 2022.

[51] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135. PMLR, 2017.

[52] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.

[53] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[54] Alexandre Galashov, Jovana Mitrovic, Dhruva Tirumala, Yee Whye Teh, Timothy Nguyen, Arslan Chaudhry, and Razvan Pascanu. Continually learning representations at scale. In *Conference on Lifelong Learning Agents*, pages 534–547. PMLR, 2023.

[55] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems (NIPS)*, 31, 2018.

[56] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

[57] Alex Gomez-Villa, Bartlomiej Twardowski, Lu Yu, Andrew D Bagdanov, and Joost van de Weijer. Continually learning self-supervised representations with projected functional regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3867–3877, 2022.

[58] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[59] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[60] Dipam Goswami, Yuyang Liu, Bartłomiej Twardowski, and Joost van de Weijer. Fecam: Exploiting the heterogeneity of class distributions in exemplar-free continual learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[61] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[62] Yiduo Guo, Bing Liu, and Dongyan Zhao. Online continual learning through mutual information maximization. In *International Conference on Machine Learning*, pages 8109–8126. PMLR, 2022.

[63] Gunshi Gupta, Karmesh Yadav, and Liam Paull. Look-ahead meta learning for continual learning. *Advances in Neural Information Processing Systems (NIPS)*, 33:11588–11598, 2020.

[64] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[65] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

[66] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[67] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision (ECCV)*, 2020.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[69] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 831–839, 2019.

[70] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

[71] Huiyi Hu, Ang Li, Daniele Calandriello, and Dilan Gorur. One pass imagenet. *arXiv preprint arXiv:2111.01956*, 2021.

[72] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

[73] Rakib Hyder, Ken Shao, Boyu Hou, Panos Markopoulos, Ashley Prater-Bennette, and Salman Asif. Continual learning via low-rank network updates, 2022.

[74] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations (ICLR)*, 2022.

[75] Paul Janson, Wenxuan Zhang, Rahaf Aljundi, and Mohamed Elhoseiny. A simple baseline that questions the use of pretrained-models in continual learning. In *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2022.

[76] Xu Ji, João F. Henriques, Tinne Tuytelaars, and Andrea Vedaldi. Automatic recall machines: Internal replay, continual learning and the brain. *CoRR*, abs/2006.12323, 2020.

[77] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[78] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[79] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*, 2016.

[80] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems (NIPS)*, 33:18661–18673, 2020.

[81] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning representations (ICLR)*, 2015.

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

[83] Hyunseo Koh, Dahyun Kim, Jung-Woo Ha, and Jonghyun Choi. Online continual learning on class incremental blurry task configuration with anytime inference. In *International Conference on Learning Representations (ICLR)*, 2022.

[84] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops (ICCV Workshop)*, pages 554–561, 2013.

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

[86] Lilly Kumari, Shengjie Wang, Tianyi Zhou, and Jeff A Bilmes. Retrospective adversarial replay for continual learning. *Advances in Neural Information Processing Systems (NIPS)*, 35:28530–28544, 2022.

[87] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

[88] Matthias De Lange, Gido M van de Ven, and Tinne Tuytelaars. Continual evaluation for lifelong learning: Identifying the stability gap. In *The Eleventh International Conference on Learning Representations*, 2023.

[89] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[90] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[91] Kibok Lee, Kimin Lee, Jinwoo Shin, and Honglak Lee. Incremental learning with unlabeled data in the wild. pages 29–32, 2019.

[92] Kuan-Ying Lee, Yuanyi Zhong, and Yu-Xiong Wang. Do pre-trained models benefit equally in continual learning? In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6485–6493, 2023.

[93] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems*, pages 4652–4662, 2017.

[94] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion*, 58:52–68, 2020.

[95] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*, 2018.

[96] Zhizhong Li and Derek Hoiem. Learning without forgetting. *Transactions of Pattern Recognition and Machine Analyses (PAMI)*, 40(12):2935–2947, 2017.

[97] Huiwei Lin, Baoquan Zhang, Shanshan Feng, Xutao Li, and Yunming Ye. Pcr: Proxy-based contrastive replay for online class-incremental continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24246–24255, 2023.

[98] Sen Lin, Li Yang, Deliang Fan, and Junshan Zhang. Trgp: Trust region gradient projection for continual learning. In *International Conference on Learning Representations*, 2021.

[99] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. When machine learning meets privacy: A survey and outlook. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021.

[100] Xialei Liu, Chenshen Wu, Mikel Menta, Luis Herranz, Bogdan Raducanu, Andrew D. Bagdanov, Shangling Jui, and Joost van de Weijer. Generative feature replay for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[101] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics training: Multi-class incremental learning without forgetting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12245–12254, 2020.

[102] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[103] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido M Van de Ven, et al. Avalanche: an end-to-end library for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3600–3610, 2021.

[104] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[105] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. 2023.

[106] Divyam Madaan, Jaehong Yoon, Yuanchun Li, Yunxin Liu, and Sung Ju Hwang. Representational continuity for unsupervised continual learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[107] Simone Magistri, Tomaso Trinci, Albin Soutif, Joost van de Weijer, and Andrew D. Bagdanov. Elastic feature consolidation for cold start exemplar-free incremental learning. In *The Twelfth International Conference on Learning Representations*, 2024.

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

[109] Zheda Mai, Ruiwen Li, Hyunwoo Kim, and Scott Sanner. Supervised contrastive replay: Revisiting the nearest class mean classifier in online class-incremental continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3589–3599, 2021.

[110] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[111] Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation. *arXiv preprint arXiv:2010.15277*, 2020.

[112] Sanket Vaibhav Mehta, Darshan Patil, Sarath Chandar, and Emma Strubell. An empirical investigation of the role of pre-training in lifelong learning. *Journal of Machine Learning Research*, 24(214):1–50, 2023.

[113] Jorge A Mendez and ERIC EATON. Lifelong learning of compositional structures. In *International Conference on Learning Representations (ICLR)*, 2020.

[114] Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *International Conference on Learning Representations (ICLR)*, 2018.

[115] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*, 2020.

[116] Sajjad Moazeni. Energy consumption of large language models: An interview with sajjad moazeni. https://www.washington.edu/news/2023/07/27/how-much-energy-does-chatgpt-use/, August 2023.

[117] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[118] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2(3):4, 2018.

[119] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pages 722–729. IEEE, 2008.

[120] Fabrice Normandin, Florian Golemo, Oleksiy Ostapenko, Pau Rodriguez, Matthew D Riemer, Julio Hurtado, Khimya Khetarpal, Ryan Lindeborg, Lucas Cecchi, Timothée Lesort, et al. Sequoia: A software framework to unify continual learning research. *arXiv preprint arXiv:2108.01005*, 2021.

[121] Oleksiy Ostapenko, Timothee Lesort, Pau Rodriguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin. Continual learning with foundation models: An empirical study of latent replay. In *Conference on Lifelong Learning Agents*, pages 60–91. PMLR, 2022.

[122] Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition. *Advances in Neural Information Processing Systems (NIPS)*, 34:30298–30312, 2021.

[123] Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. Deep learning for financial applications: A survey. *Applied soft computing*, 93:106384, 2020.

[124] Aristeidis Panos, Yuriko Kobe, Daniel Olmeda Reino, Rahaf Aljundi, and Richard E. Turner. First session adaptation: A strong replay-free baseline for class-incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 18820–18830, October 2023.

[125] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[126] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink, 2022.

[127] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1406–1415, 2019.

[128] Michael P Perrone and Leon N Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In *How We Learn; How We Remember: Toward An Understanding Of Brain And Neural Systems: Selected Papers of Leon N Cooper*, pages 342–358. World Scientific, 1995.

[129] Grégoire Petit, Adrian Popescu, Hugo Schindler, David Picard, and Bertrand Delezoide. Fetril: Feature translation for exemplar-free class-incremental learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3911–3920, 2023.

[130] Ameya Prabhu, Hasan Abed Al Kader Hammoud, Puneet K Dokania, Philip HS Torr, Ser-Nam Lim, Bernard Ghanem, and Adel Bibi. Computationally budgeted continual learning: What does matter? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3698–3707, 2023.

[131] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European Conference on Computer Vision (ECCV)*, pages 524–540. Springer, 2020.

[132] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[133] Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations*, 2022.

[134] Amal Rannen, Rahaf Aljundi, Matthew B Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1320–1328, 2017.

[135] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2001–2010, 2017.

[136] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[137] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, , and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning representations (ICLR)*, 2019.

[138] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[139] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[140] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2020.

[141] Laine Samuli and Aila Timo. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[142] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[143] Mert Bulent Sariyildiz, Yannis Kalantidis, Karteek Alahari, and Diane Larlus. No reason for no supervision: Improved generalization in supervised models. In *International Conference on Learning Representations (ICLR)*, pages 1–26, 2023.

[144] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

[145] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[146] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[147] Albin Soutif-Cormerais, Antonio Carta, Andrea Cossu, Julio Hurtado, Vincenzo Lomonaco, Joost Van de Weijer, and Hamed Hemati. A comprehensive empirical evaluation on online continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3518–3528, 2023.

[148] Albin Soutif-Cormerais, Antonio Carta, and Joost Van de Weijer. Improving online continual learning performance and stability with temporal ensembles. In *Conference on Lifelong Learning Agents*, pages 828–845. PMLR, 2023.

[149] Albin Soutif-Cormerais, Marc Masana, Joost Van de Weijer, and Bartłomiej Twardowski. On the importance of cross-task features for class-incremental learning. In *International Conference on Machine Learning Workshop*, 2021.

[150] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Advances in neural information processing systems (NIPS)*, 30, 2017.

[151] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[152] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. In *NeurIPS Continual Learning Workshop*, 2018.

[153] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *Annual Conference on Neural Information Processing Systems (NIPS)*, 2016.

[154] Zhen Wang, Liu Liu, Yajing Kong, Jiaxian Guo, and Dacheng Tao. Online continual learning with contrastive vision transformer. In *European Conference on Computer Vision*, pages 631–650. Springer, 2022.

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

[156] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.

[157] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.

[158] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[159] Martin Wistuba, Lukas Balles, Giovanni Zappella, et al. Continual learning with low rank adaptation. In *NeurIPS 2023 Workshop on Distribution Shifts: New Frontiers with Foundation Models*, 2023.

[160] Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 11217–11227. PMLR, 18–24 Jul 2021.

[161] Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. *Annual Conference on Neural Information Processing Systems (NIPS)*, 31:5962–5972, 2018.

[162] Tz-Ying Wu, Gurumurthy Swaminathan, Zhizhong Li, Avinash Ravichandran, Nuno Vasconcelos, Rahul Bhotika, and Stefano Soatto. Class-incremental

learning with strong pre-trained models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9601–9610, 2022.

[163] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 374–382, 2019.

[164] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. *arXiv preprint cmp-lg/9406033*, 1994.

[165] Lu Yu, Bartlomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6982–6991, 2020.

[166] Gengwei Zhang, Liyuan Wang, Guoliang Kang, Ling Chen, and Yunchao Wei. Slca: Slow learner with classifier alignment for continual learning on a pre-trained model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19148–19158, October 2023.

[167] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE Workshop on Applications of Computer Vision*, pages 1131–1140, 2020.

[168] Yaqian Zhang, Eibe Frank, Bernhard Pfahringer, Albert Bifet, Nick Jin Sean Lim, and Alvin Jia. Closed-loop control for online continual learning, 2022.

[169] Yaqian Zhang, Bernhard Pfahringer, Eibe Frank, Albert Bifet, Nick Jin Sean Lim, and Alvin Jia. A simple but strong baseline for online continual learning: Repeated augmented rehearsal. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems (NIPS)*, 2022.

[170] Da-Wei Zhou, Fu-Yun Wang, Han-Jia Ye, and De-Chuan Zhan. Pycil: a python toolbox for class-incremental learning. *SCIENCE CHINA Information Sciences*, 66(9):197101–, 2023.

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

[172] Fei Zhu, Zhen Cheng, Xu-yao Zhang, and Cheng-lin Liu. Class-incremental learning via dual augmentation. *Advances in Neural Information Processing Systems*, 34:14306–14318, 2021.

[173] Fei Zhu, Xu-Yao Zhang, Chuang Wang, Fei Yin, and Cheng-Lin Liu. Prototype augmentation and self-supervision for incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5871–5880, 2021.

[174] Kai Zhu, Wei Zhai, Yang Cao, Jiebo Luo, and Zheng-Jun Zha. Self-sustaining representation expansion for non-exemplar class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9296–9305, 2022.