ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Continual Learning for Computer Vision Applications

Presentata da
Dott. Lorenzo Pellegrini

Supervisore
Prof. Davide Maltoni

Dottorato di Ricerca in

Computer Science and Engineering

Ciclo XXXIV

Coordinatore Dottorato

Prof. Davide Sangiorgi

Settore Concorsuale
09/H1

Settore Scientifico Disciplinare
ING-INF/05

ESAME FINALE ANNO 2022

Abstract

Continual Learning for Computer Vision Applications

Lorenzo Pellegrini

One of the most visionary goals of Artificial Intelligence is to create a system able to mimic and eventually surpass the intelligence observed in biological systems including, ambitiously, the one observed in humans. The main distinctive strength of humans is their ability to build a deep understanding of the world by *learning continuously* and drawing from their experiences. This ability, which is found in various degrees in all intelligent biological beings, allows them to *adapt* and properly react to changes by incrementally expanding and refining their knowledge. Arguably, achieving this ability is one of the main goals of Artificial Intelligence and a cornerstone towards the creation of intelligent artificial agents.

Following a "cold era" during which the interest in the Artificial Intelligence field was greatly reduced, a renewed interest in approaches based on Neural Networks allowed this field to shine once again. Modern Deep Learning approaches allowed researchers and industries to achieve great advancements towards the resolution of many long-standing problems in areas like (and not limited to) Computer Vision and Natural Language Processing. However, while this current age of renewed interest in AI allowed for the creation of extremely useful applications, a concerningly limited effort is being directed towards the design of systems able to learn continuously.

Current AI systems, due to their structural and algorithmic design, fail at adapting to changes in a meaningful way. Moreover, they fail at learning how to address problems different from the ones for which they have been built or trained to solve. As of today, there is a lot of progress to be made to fill the gap that separates artificial and biological intelligence.

The biggest problem that hinders an AI system from learning incrementally is the *catastrophic forgetting* phenomenon. This phenomenon, which was discovered in the 90s, naturally occurs in Deep Learning architectures where classic learning paradigms are applied when learning incrementally from a stream of experiences. If the system can't access previous training data, then the knowledge model collapses erasing all previous knowledge. Forgetting happens because classic learning paradigms cannot adapt to continuous shifts in the data distribution, which is something biological systems excel in handling.

My PhD work revolved around the *Continual Learning* field, a sub-field of Machine Learning research that has recently made a comeback following the renewed interest in Deep Learning approaches. The visionary promise of the continual learning field is to develop novel approaches that can allow intelligent artificial systems to incrementally expand, refine and adapt their knowledge by learning from experiences in a human-like way. This dissertation will focus on a comprehensive view of continual learning by considering algorithmic, benchmarking, and applicative aspects of this field. This dissertation will also touch on community aspects such as the design and creation of research tools aimed at supporting Continual Learning research, and the theoretical and practical aspects concerning public competitions in this field.

Outline

Chapter 1 introduces the concepts and motivations regarding the Continual Learning field. In particular, the strengths and weaknesses of classic Deep Learning approaches are analyzed to pinpoint the major research challenges. Building upon that, an introduction about the open issues and the mainstream research directions is given, which serves as the reference point for the rest of the dissertation.

Chapter 2 describes the research efforts in improving the realism of continual learning scenarios and benchmarks. The chapter opens with a description of the most frequently used benchmarks and protocols along with a summary of the terminology and assumptions found in the literature. Following a discussion about the main issues found in mainstream benchmarks, the NICv2 protocol is presented, which is used as the reference family of benchmarks throughout the thesis.

Chapter 3 introduces continual learning techniques (*AR1** and *Latent Replay*) able to handle complex scenarios. In particular, algorithmic contributions are focused on the creation of efficient algorithms able to handle the realistic scenarios described in the previous chapter.

Chapter 4 revolves around the libraries and frameworks aimed at supporting the research efforts in the continual learning field. The surge of interest in Continual Learning is very recent and no common methodology and codebases were accepted through the research community. Following an analysis of the mainstream research directions and common practices, a list of desiderata for a continual learning library is determined. Then, building upon those, the design and implementation of the *Avalanche* library are described.

Chapter 5 is focused on practical applications of continual learning. Applications for resource-constrained devices are described by showing how the findings outlined in Chapter 3 can be applied in practice. In particular, this chapter details the design and implementation of different practical applications including the ones that originated from close collaborations with both external research groups and industry partners. Moreover, a dissertation on theoretical and practical aspects of running and participating in public Continual Learning competitions is proposed.

Finally, conclusions and future challenges in the continual learning field are discussed in Chapter 6.

Publications

- *Latent Replay for Real-Time Continual Learning*, published at the *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2020.*
- *Rehearsal-Free Continual Learning over Small Non-IID Batches*, published at the *1st CLVision Workshop, Conference on Computer Vision and Pattern Recognition, 2020.*
- *Memory-Latency-Accuracy Trade-Offs for Continual Learning on a RISC-V Extreme-Edge Node*, published at the *IEEE International Workshop on Signal Processing Systems, 2020.*
- *Continual Learning at the Edge: Real-Time Training on Smartphone Devices*, published at the *European Symposium on Artificial Neural Networks, 2021.*

- *Avalanche: an End-to-End Library for Continual Learning*, published at *2nd CLVision Workstop, Conference on Computer Vision and Pattern Recognition, 2021*
- *CVPR 2020 Continual Learning in Computer Vision Competition: Approaches, Results, Current Challenges and Future Directions*, published in the *Artificial Intelligence, Elsevier* journal, 2021.
- *IROS 2019 Lifelong Robotic Vision: Object Recognition Challenge*, published in the *IEEE Robotics & Automation Magazine* journal, 2020.
- *Is Class-Incremental Enough for Continual Learning?*, published in the *Frontiers in Artificial Intelligence* journal, 2022.
- *A Weakly Supervised Approach for Recycling Code Recognition*, 2021. Submitted as a journal paper.
- *Generative Negative Replay for Continual Learning*[1], 2021. Submitted as a conference paper.

[1]Not discussed in this thesis.

Acknowledgements

I would like to thank my supervisor Prof. Davide Maltoni for his guidance and support during my PhD. His passion for Artificial Intelligence inspired me in adventuring in tough but very rewarding projects. Thanks to him I got to explore the extremely interesting practical aspect of the Continual Learning field, an aspect that was receiving little to no attention from the community. I am especially thankful for the trust and the constant support he gave me in persecuting the most variegated projects during these years.

A special thank goes to Prof. Dario Mario and Prof. Matteo Golfarelli for the support provided as members of the internal evaluation board.

During the last year of my PhD I got the fantastic opportunity to work in the very energetic environment of Facebook AI Research. My most sincere gratitude goes to my FB research supervisor Abhijit Ogale, who provided great guidance during the internship, and to my colleague Fu-Jen Chu, who assisted me with all day-to-day issues.

There have been many ups and downs in these three years. The pandemic has been very demanding in terms of mental stress and it posed a serious risk in terms of research productivity. I would not have been able to achieve the results here discussed without the support of my colleagues at the SmartCity-BioLab laboratory in Cesena: a special thank goes to Vincenzo Lomonaco, Gabriele Graffieti, and Guido Borghi. Our day-to-day discussions covering both research and frivolous elements have been of utmost importance during these years.

I am also very proud to be part of the Avalanche family. The Avalanche project has been a turning point during my PhD: working on such a visionary and useful project for the community allowed me to get back on track after the most demanding parts of the pandemic. I want to thank Antonio Carta and Andrea Cossu for the great guidance and effort they put into this.

A *prosit!* to all my friends that supported and put up with me. These years would not have been mentally manageable without them.

Finally, this thesis is dedicated to my family. I want to thank them for all the support they gave me during these years. I truly have been blessed to have such an awesome family that never doubted my abilities and that always assisted me in times of need.

Contents

List of Figures

List of Tables

Chapter 1

Background and Motivation

Artificial Intelligence has always been one of the most active fields of Computer Science, and also one of the most popular among the general public. The field comprehends many different research directions, but all of them are directed towards the common goal of reaching that scientific advancement that could allow for the creation of autonomous intelligent agents. Such agents, in the collective imagination, are usually linked to the idea of robotic or personal assistant systems. Many artificial intelligent systems and applications exist nowadays, but none of them feature that kind of autonomy and adaptability found in humans. In fact, the field is still very far from formulating a solution able to mimic the most relevant feature of biological systems: the ability to learn over time. Among the most evident strengths of human beings is their ability to *learn from experiences*, *continuously*, *autonomously*, and during their *whole lifetime*. With this innate ability, they can build an internal knowledge model able to describe the world around them. This also applies to other intelligent animals but, in both the collective imagination and among researchers, the most visionary goal is to reach the degree of adaptability and intelligence found in humans.

As of now, the main issue that is preventing this kind of intelligence to be reached is exactly the lack of ability to adapt over time. The world is a very complex environment in which *continuous change is the normality* and *exceptions are everyday events*. Many researchers started to work on the idea of enabling *continuous learning* capabilities in artificial systems. This new research direction, also described by the *lifelong learning* and *incremental learning* terms, is the focus of this thesis. Of course, intelligence cannot be defined by the ability to adapt during time alone. However, this "feature" found in biological systems constitutes part of the foundations of Artificial General Intelligence.

Researchers that first started to explore this field (in the 1990s) were immediately welcomed by the same obstacle that is blocking the path nowadays: the *catastrophic forgetting* phenomenon. This phenomenon, described in more detail later, happens when trying to learn new concepts in an incremental (sequential) way. This problem was immediately diagnosed in deep models trained with gradient optimization, which constitute the mainstream structure and technique used in modern artificial intelligence.

1.1 Modern Artificial Intelligence

While artificial intelligent systems still lack the ability to learn continuously, modern artificial intelligence techniques were able to solve or, at least, reduce the gap with human performance on many complex problems. As of today, the most popular approaches are *deep learning* ones. Works in the continual learning area make use of deep learning models as the main mechanism to simultaneously learn representations and accumulate knowledge. Before diving into the discussion of the continual learning field, it is worth spending a few words about the popularity, strengths, and limits of deep learning approaches.

1.1.1 Strengths of Deep Learning approaches

The use of deep learning techniques is now considered the main approach to be employed when solving many long-standing problems in areas such as computer vision, natural language processing, signal processing, etcetera. Deep learning is based on the idea of stacking a deep architecture/hierarchy of components (layers). In practice, deep neural networks, composed of a stack of tens of layers, are used to learn relevant representations from huge amounts of data. The use of deep learning techniques popularized the made the concept of learning in an *end-to-end* fashion.

The ability of this approach to *learn relevant representations* is arguably the main strength of deep learning approaches. Representations are learned by using a gradient-based optimization which is applied in depth through the model architecture. From the representation learning point of view, the desirable features that made deep learning popular are:

- **Learning from high-dimensional data directly**. This is arguably the biggest strength: each problem to solve is different, which means that a technique able to learn problem-specific representations autonomously is highly desirable. On the opposite, classic techniques usually exploit manually engineered features, which are problem-specific and not generalizable across tasks. Although an in-depth knowledge of the problem to solve (or at least the macro-area it belongs to) is important, a system able to learn representations directly from data allows for researchers (and engineers) to not focus on details regarding the problem at hand, making it easier to develop and release new models. This is particularly important in situations in which data is high-dimensional, which happens in most real-world vision and NLP applications.
- **Simple learning technique**. In deep neural networks, the goal of the learning technique is to minimize a loss function that represents the discrepancy between the output and the expected result. This idea is then applied in practice by applying iterative gradient descent steps, in which the gradient of the loss function is computed through the network architecture by backpropagation. Of course, there are differences between different ways to do this (although the Stochastic Gradient Descent is commonly used), but the same idea is shared across the whole deep learning area. No matter the problem to solve and the macro-area it belongs to, the promise of deep learning is that this simple technique can be applied to learn relevant representations.
- **Exploiting data abundance**. Not only deep learning techniques can extract representations from the data directly, but they are also able to exploit the abundance of data in the training dataset. One of the most iconic ideas around modern artificial intelligence is that the performance of the resulting model is related to the amount (and quality) of data available in the training dataset. In other words, it is better to add more training data than to spend time designing complex learning techniques. With this idea in mind, following the renaissance

of neural network techniques, companies (and sometimes universities) started to gather and publish bigger and bigger datasets for the most disparate tasks.

Neural network techniques have not always been that popular. The first scientists that worked in this direction stumbled upon obstacles that blocked research advancements. Firstly, early works focused on shallow models, which could not properly be used to learn complex representations from data. Secondly, the lack of abundant computational power, which is now provided by modern GPUs and ad-hoc accelerators, made it difficult to train bigger architectures. Finally, the use of MLPs alone instead of domain-specific architectures (CNNs, RNNs, Transformers), made it difficult to approach problems featuring high-dimensional data.

In fact, one of the most popular events that made brought neural network techniques popular again is the use of a CNN [82] to drastically reduce the error on the popular ImageNet challenge [33]. At that point, it was clear that deep learning techniques could enable efficient representation learning (due to the aforementioned elements: using high-dimensional data, using a simple training technique, and exploiting the abundant training data found in ImageNet). From that moment, a revolution within both research and industry began. Nowadays, a lot of commonly used applications are backed by one or mode deep learning models: photo enhancers and organizers, text and speech translators, quality assurance and anomaly detection systems, biometric systems, etcetera.

For a more in-depth overview of the field, we recommend referring to popular books and articles on the argument: [49, 85].

1.1.2 Current limits of Neural Network-based intelligent systems

Lack of adaptability The most evident issue of DNNs is the lack of mechanisms that could be used to allow for a proper adaptation of the knowledge. In particular, *DNNs are not able to adapt and generalize to new circumstances* and environments. To enable continual learning capabilities, an intelligent agent should be able to contextualize and specify already learned behaviors but also face never seen scenarios. To do so, the agent has to learn new skills and concepts, or even just incrementally improve how it behaves when facing already encountered problems and environments. Alas, deep learning and gradient-based optimization techniques are not sufficient to allow for this kind of adaptability due to the *catastrophic forgetting* phenomenon. When faced with new concepts to learn or drifts in data distribution, a model trained with classic gradient-based techniques forgets all previously accumulated knowledge (more on this in Section 1.2).

Low efficiency Another limit of DNNs is the *computational power (and time) needed to train deep architectures*. While the general availability of powerful accelerators, such as GPUs and specialized hardware, allows for huge models to be trained, the training process is still too computational and energy-expensive w.r.t. biological systems. Not only a higher efficiency is desirable for server-side applications, but they are necessary to enable continual learning capabilities in edge devices such as robots and embedded systems.

Dealing with small and non-i.i.d. batches One of the aspects that make deep learning techniques so powerful is their ability to exploit large amounts of data. However, when the amount of data is too limited, learning useful representations may become impossible. Dealing with small batches of data is common in continual learning scenarios. In addition, while training deep learning models through SGD-based optimization techniques, the mini-batches that constitute the stream of training data are usually obtained by shuffling and splitting the training dataset. In continual learning scenarios, data may become available during time as non-i.i.d. batches. Training using this batch composition quickly leads to knowledge destruction in the model, which mandates CL-specific techniques to be employed.

Lack of structural adaptability Given a problem to be solved, a network architecture is defined (usually borrowed from mainstream ones) and the training loop is optimized by selecting the optimization technique and hyperparameters. This means that the architecture itself is fixed throughout the whole training process. Architecture search techniques [37] try to compensate for this limit. However, even in that case, the search is done in the context of a classic training scheme and it can't be applied to continual learning scenarios.

1.2 On Continual Learning

The efforts toward enabling continual learning in intelligent systems are based on the simple idea of learning over time. More precisely, the idea is to design an artificial system that can, at the same time: i) learn new concepts, ii) preserve previously acquired knowledge (and forget unuseful information in a controlled way), iii) reuse knowledge (generalize) across tasks.

Reaching the kind of intelligence found in biological intelligent systems, that constitute proof of work, can be considered the main goal. Biological systems, like humans, can learn new concepts while retaining existing knowledge, even when abrupt drifts occur in the data distribution. In biological systems, A sudden complete loss of previously acquired knowledge is considered an anomaly. These systems learn concepts sequentially one after another by being exposed to a stream of sensory information. Alas, the forgetting phenomenon makes it impossible to handle such a learning scheme in artificial systems. Early works in the field (Section 1.2.2) show that the interference problem is related to how gradient-based optimization operates, which is not able to directly retain already acquired knowledge.

Once aware of these problems, the research community deemed it necessary to move in the direction of developing ad-hoc techniques to contrast forgetting. Because of that, the continual learning field revolves around the definition of designing novel benchmarks and techniques that could bring the artificial intelligence field closer to the goal of building general artificial intelligent systems.

1.2.1 Goals and Motivations

The concept of learning continually from experience has always been present in artificial intelligence and robotics [173, 181]. This idea has also been explored in research names with different names, such as *Lifelong Learning* [25, 172] and *Incremental Learning* [48, 135]. All these directions focus on the idea of learning over time and, considering their broad landscape, they may be considered as the same field as well.

To reach that kind of adaptability in time, the *stability/plasticity dilemma* [53] must be solved. In practice, biological systems can accommodate new knowledge (plasticity) but, at the same time, they can preserve previously acquired knowledge (stability).

It is clear that deep learning models, in which learnable parameters are trained to minimize a loss function via gradient-based mechanisms, are inherently too plastic to be able to retain previous knowledge.

One of the main reasons for pushing research into the field of continual learning is that continual learning could be a fundamental element in empowering intelligent embodied agents. Such agents would operate in a realistic environment and they should be autonomous. An example of the real need for a continual learning system can once again be made in the context of robotics [172]. Real-world embedded robots are in strong need of learning over time, specializing and adapting their behaviors locally in an efficient way [171]. This means that the underlying continual learning system should be able to:

- *Learn with limited (or none) labels.* The ability to learn autonomously can be considered the most fundamental element of autonomous intelligent systems. Learning without supervision is common for biological systems thanks to their innate ability to explore autonomously (curiosity) and reason about the environment.
- *Operate without task labels/boundaries.* Just like with problem-specific labels, the need for some kind of boundary or other signal describing the task at hand would hinder the autonomy of the system, making it terribly different from a biological system.
- *Handle unexpected scenarios/tasks/problems.* The system should be able to handle unexpected scenarios and tasks. A system that can only operate in a specific setting is too limited and it will be impossible to learn and act autonomously and effectively.
- *Learn from sensory information (online and streaming).* Biological systems learn from a continuous stream of data coming from their "sensors". Apart from being excellent proof-of-works, they are also very efficient in doing so. In other words, CL systems should be able to quickly and efficiently learn from streams instead of batches of data. This becomes difficult when facing high-dimensional data, which is the kind of data biological systems normally handle.
- *Learn with constant resources.* Resources include all "hardware" elements such as memory, computational power, and energy consumption. This, coupled with the fact that learning should be done in a "streaming" fashion, poses strict constraints on the ideal continual learning solution.
- *Forget when necessary.* This feature has been overlooked in the current continual learning literature. While most of the literature focuses on solving the catastrophic forgetting problem, as far as we know, no efforts toward designing a controlled forgetting mechanism have been made so far. However, the ability to gracefully forget past knowledge is very well-known in humans and a desirable feature for embodied intelligent agents, too.

That said, the scope of application of continual learning techniques may be narrower in scope: systems able to autonomously learn how to detect new classes of objects, defects, voices, etcetera could all benefit from advancements in the continual learning field.

In addition, continual learning could empower AutoML systems [62]. Without continual learning capabilities, these systems have to retrain the model on the whole available data stored over time. This has a strong impact on their ability to quickly adapt to changes and on their use of energy and computational resources. Considering that AutoML is becoming more and more relevant in the industry, and also considering how much expensive they are, the use of effective continual learning techniques could revolutionize the way these services are managed and provided.

1.2.2 Brief history

The idea of learning continually from experience is one of the fundamental topics in artificial intelligence and robotics since their birth [173, 181]. However, it is only at the end of the 20th century that it has begun to be explored more systematically.

Research on the possibility of continually training a deep learning model dates back to 1989 when McCloskey and Cohen [107] proposed the initial experiments able to show the inefficacy of the classic training methods. In particular, they identify the *catastrophic interference* issue, which is the most studied problem in continual learning literature so far. The phenomenon resulting from that interference is usually referred to as *catastrophic forgetting*. Since then, many works were directed toward the study of methods that could enable the continuous learning of new concepts. In particular, Turn and Mitchell [172] popularized the idea of lifelong learning on classification tasks, while Ring [140] studied the idea of applying continual learning to reinforcement learning tasks. These two research directions are arguably the most active ones even in recent times.

Starting from these early works, it was clear that most of the work would go in the direction of using neural network models as the main learning drivers: the use of neural networks enable meaningful representations to be learned and shared across sequentially learned tasks. In fact, using manually engineered representations would make it impossible to continually learn novel, possibly extremely varied, concepts. At the same time, no matter the specific setting (supervised classification, reinforcement learning, . . .), it was evident that specific approaches were required to limit the catastrophic inference phenomenon.

In addition, it is at that time that many assumptions and concepts (that are now consolidated in the literature) were posed. Common assumptions include the idea of not being able to store a massive amount of data from past experiences, the idea of limiting the memory and computational overhead introduced by learning a substantial number of concepts, and the idea of keeping the training and inference time as much constant as possible. These are all desirable elements that are needed for enabling continual learning on robotic systems and autonomous agents in general. In fact, other early works were focused on strategies able to mitigate the forgetting by not storing elements from the training data at all. This constraint has been relaxed in more recent works and replay strategies are now widely used (more on this in Section 1.3.1). However, the idea of pseudo-rehearsing was already proposed [142]. In general, initial works focused on multi-task setups at a small scale of few examples and considering small and shallow networks [83, 8].

Given these early works, it was clear that the most prominent macro-directions were *continual supervised learning* and *continual reinforcement learning*, which are both very akin to the robotics and neurobiology fields. This generally holds even in recent times. Apart from these two directions, it is worth noting that *continual unsupervised learning* [134, 2] and *continual semi-supervised learning* [154, 179] have been less explored. Continual learning in an unsupervised or semi-supervised way is highly desirable and it is an open challenge (more on this in Section 1.3.2).

Continual Supervised Learning In the context of continual learning, most efforts have historically been directed towards fully supervised learning contexts. Prominent efforts include [171], where each new task consists in learning a concept using binary classification. Several CL techniques were then proposed in the contexts of memory-based learning and artificial neural networks while remaining within the idea of not storing past data. Among these, it is worth noting pseudo-rehearsal techniques proposed in [142, 161, 162]. More recent works include "Efficient lifelong learning algorithm" (ELLA) (proposed in [148]), which improves the multi-task learning method proposed by [84]. Here the learning tasks are independent of each other and a regularization strategy is proposed. A more theoretical study of continual learning was firstly accomplished by [123] within the PAC-learning framework. Finally, a recent review covering more recent efforts has been proposed by Parisi et al. [115], which also includes useful pointers for the *continual reinforcement learning* area.

Continual Reinforcement Learning In [111] an algorithm for robot learning was proposed that tried to capture the invariant knowledge about each task. In the doctoral thesis from M.B. Ring [140], a continual learning agent is proposed with the idea of gradually solving complicated tasks by learning easy tasks first. It is from [167] that each environment started to be treated as a separate task for continual reinforcement learning. This paradigm has been highly influential and many recent works, such as [79, 147, 98], follow the same "environment/game-as-a-task" idea. A hierarchical Bayesian continual reinforcement learning method in the framework of Markov Decision Process (MDP) has been proposed in [182], while [42] specifically worked on policy reuse in a multi-task setting. A nonlinear feedback policy that generalizes across multiple tasks was proposed in [30]. Following the idea presented with ELLA [148], a policy gradient efficient continual learning algorithm was proposed in [7]. This work was further enhanced with cross-domain continual reinforcement learning [6] and with constraints for safe continual reinforcement learning [5].

Regarding the overlap of *reinforcement learning* and *continual learning*, both areas try to tackle scenarios in which learning happens sequentially, which means that abrupt concept drifts may hinder the learning process. In addition, continual learning techniques like experience replay, regularization, etcetera (more on Section 1.3.1) can be used in both supervised and reinforcement learning. However, benchmarks are quite different, with continual reinforcement learning efforts usually involving real robotic systems or, more common in *continual reinforcement learning*, simulated environments such as Atari and other games [79, 98, 14].

A more in-depth overview of the continual reinforcement learning area can be found in [77].

1.3 State of the art

Continual learning research is moving fast and many novel techniques, benchmarks, and even whole new research directions are being actively proposed. This section is meant to describe the state of the art in terms of approaches (techniques) and long-standing challenges of the continual learning field in computer vision. In particular, the mainstream *continual supervised learning* approaches are analyzed here, while the description of commonly used benchmarks, which is a fundamental element needed to describe the current state of the art, is proposed in more detail in Chapter 2.

1.3.1 Mainstream approaches

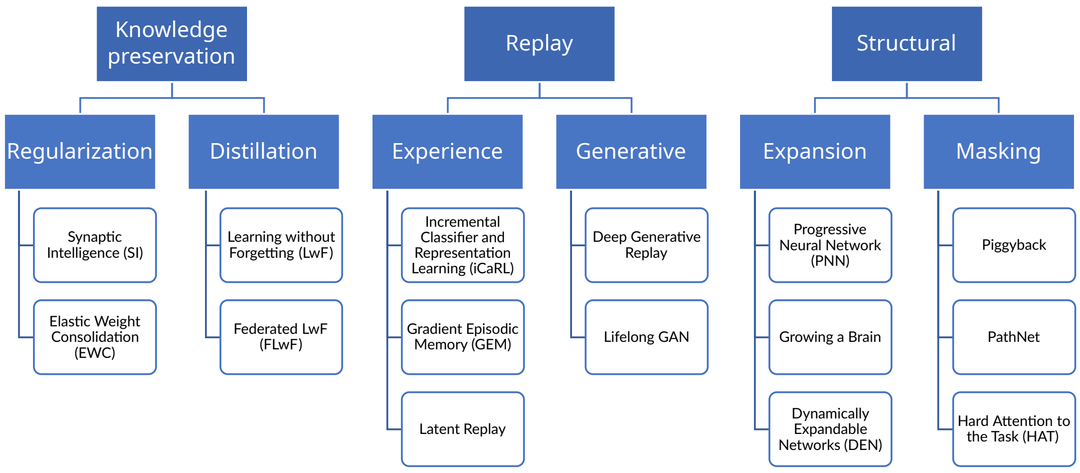
Knowledge preservation
Replay
Structural
Regularization
Distillation
Experience
Generative
Expansion
Masking
Synaptic Intelligence (SI)
Elastic Weight Consolidation (EWC)
Learning without Forgetting (LwF)
Federated LwF (FLwF)
Incremental Classifier and Representation Learning (iCaRL)
Gradient Episodic Memory (GEM)
Latent Replay
Deep Generative Replay
Lifelong GAN
Progressive Neural Network (PNN)
Growing a Brain
Dynamically Expandable Networks (DEN)
Piggyback
PathNet
Hard Attention to the Task (HAT)

FIGURE 1.1: Proposed classification of mainstream continual supervised learning techniques. For each subdivision, a few reference techniques are listed.

Although it is a particularly recent area, continual learning features a lot of research directions and areas of applications. Because of this and due to the complexity of the problems it aims to solve, a lot of different techniques to tackle the forgetting problem are being proposed. Techniques are very different from each other, and only a few common elements can be pointed out. While a formal taxonomy is hard to define, a very high-level, non-exhaustive, coarse categorization of mainstream continual learning techniques is here provided by the means of three macro-categories (also outlined in Figure 1.1):

- Knowledge preservation techniques (regularization and distillation)
- Replay techniques (including generative approaches)
- Structural techniques (model expansion and masking)

This classification based on these three categories is well known in the literature [115, 103] (although different names can be attributed to each category). This way to categorize continual learning approaches is based on the analysis of the strategies employed in the earliest academic works in the field. We argue that, despite its simplicity and the fact that it may not cover all possible approaches, this organization is still valid and can categorize even more recent strategies, which usually feature a hybridization of techniques taken from more than one of those categories. This hybridization is required as techniques from each category try to limit the forgetting issue each in a different way and, most importantly, these techniques are usually not mutually exclusive.

That said, given the aforementioned subdivision, it is worth spending a few words on the mainstream approaches found in the literature for each category. This is useful to delineate the strength and weaknesses of each macro-approach, as well as their extremes.

Knowledge preservation techniques

Knowledge preservation techniques try to tackle the catastrophic forgetting phenomenon by limiting the knowledge drift that happens in deep neural networks when training on a new batch of data. Techniques belonging to this category do not make use of an external replay buffer, which means that using these techniques may have positive effects in terms of privacy preservation and persistent memory usage. However, it must be noted that some additional memory may still be necessary to store model-related additional data (which may happen in regularization techniques). This particularly popular category of techniques can be split into two sub-categories: regularization and distillation techniques.

Regularization Regularization techniques try to limit the knowledge drift by limiting (slowing down) the update rate of learnable parameters in the model. Regularization techniques are usually based on some definition of *parameter importance*. Important parameters are preserved from abrupt updates while less important ones are usually left freer to learn. Elements such as how the *importance* is measured and bookmarked for each parameter, and the way parameter updates are regulated based on that importance, are specific to each technique.

It is useful to pin-point a couple of techniques to allow for better framing of the regularization techniques. Synaptic Intelligence (SI) [189] and Elastic Weight Consolidation (EWC) [79] are well-known continual learning strategies based on slightly different mechanisms. I chose to bring these two examples as these techniques have been presented by the respective authors as stand-alone, which means that they are pure regularization strategies.

In EWC, the importance of each weight is given by the diagonal of the Fisher information matrix. In SI, the importance of each parameter is measured as its contribution (during all experiences so far) in decreasing the value of the target loss function. The idea in both techniques is that important parameters retain knowledge from past experiences while non-important ones may carry less knowledge and so they should be left freer to learn.

The mechanism regulating the slowdown of parameter updates is similar in both mechanisms: the loss function guiding the optimization is integrated with an importance-aware component that penalizes the change of learnable parameters. This component is directly proportional to the importance of each parameter. A different update mechanism is described in [103] that, starting from SI, proposes a simplified framework for regularization techniques, in which the importance is used to alter the learning rate of each parameter instead of altering the loss function.

Distillation Distillation techniques try to limit the forgetting phenomenon by distilling the knowledge of a previous model (usually the one at $t-1$) so that the model obtained when incrementally trained on the new experience may carry both the past and new knowledge. The idea of distilling knowledge from one source model to another one is well known in classic deep learning literature [65].

Learning Without Forgetting (LwF) was introduced in [92] and is arguably the most known technique in this category. In LwF, before the beginning of the training phase, training samples from the current experience are forwarded in the model to record the output from heads associated with previous tasks. Then, while training on the current experience, the loss function is integrated with a component penalizing changes in the output of previous tasks. In addition, LwF has been used in the context of continuous federated learning, such as in FLwF [174]. In general, most continual learning distillation-based techniques are inspired by the successful approach of LwF.

Replay techniques

Replay techniques (sometimes named *rehearsal techniques*) are widely used blocks of modern continual learning strategies. This category of techniques tries to limit the forgetting issue by replaying data that should be representative of previous experiences. This forces the learning algorithm to optimize the loss for old instances, too. This also prevents a situation in which the model is optimized on new concepts without old ones used to counterbalance the learning process, which may result in catastrophic forgetting. This is a very simple idea and even simple approaches lead to excellent results in terms of retained knowledge. On the other hand, replay techniques usually require abundant persistent storage memory to store the *replay buffer* or the *generative model(s)*. As such, this family of techniques can be divided into two sub-categories: *experience replay* and *generative replay*.

Experience replay Experience replay techniques are based on the idea of keeping a replay buffer in which to store instances from previous experiences. Instances are then jointly replayed with data from the current batch. This has three implications: i) it may pose a problem in terms of used storage memory, ii) replaying instances requires additional computational resources (or time) to complete the training phase, iii) storing raw data may not be possible due to privacy constraints. Despite these issues, experience replay techniques are widely used as they can be easily used in conjunction with knowledge preservation and structural techniques.

The recipe to design a replay technique usually involves defining the following elements:

- Fixed-size or expanding buffer. Many works follow the idea of keeping a fixed-size replay buffer as this should better model a situation in which the persistent memory cannot be easily expanded to accommodate for more instances. However, other works expand the buffer depending on the unique concepts (classes) learned up to that moment.
- Instances selection and replacement. This is arguably the most important component in a replay strategy as it defines most of the effectiveness. No matter if a fixed-size or an expanding buffer is employed, it is possible to store only a limited amount of instances. This means that a mechanism to select new instances to be inserted in the buffer must be designed. The same mechanism is usually in charge of removing instances already in the buffer if the maximum capacity has been exceeded. This usually includes balancing the replay instances in the buffer (e.g., class-balanced), which is usually desirable.
- Training-time sampling. Most works use the whole replay buffer by mixing all old and new instances. A more sophisticated approach (especially common

in streaming learning scenarios) is to use only a limited subset of the replay buffer to limit the performance impact.

- Old and new instances mixing. The simpler solution is to randomly shuffle new and replay instances as if they were a single training dataset. More sophisticated approaches may be desirable when using relatively small replay buffers, such as fixing the number of replay instances to be inserted in each minibatch. In addition, some approaches, like GEM [99], use replay exemplars in a different way.

Class-balanced replay with random instances insertion/replacement is arguably the simpler and most used solution. However, more complex solutions have been proposed, such as the herding selection mechanism proposed in iCaRL [135], which is now quite popular in literature. Other well-known strategies, like GEM [99] and A-GEM [20], use experience replay to limit the degree of forgetting on past tasks by the means of gradient projection.

Experience replay is not strictly linked to replaying raw data as-is. For instance, the latent replay mechanism is proposed in Section 3.2, which uses latent activations instead of raw images.

Generative replay In generative replay techniques, instead of replaying previous instances, one or more generative models are used to generate data able to represent the data distribution from past experiences. As with experience replay, generative replay incurs storage-related issues due to the space required to store the generative model(s). In addition, replay instances must be generated, which means that the generative process will take computational resources and time. These techniques are very interesting for three reasons: i) generative models, depending on how the architecture and how they are used, may be less prone to forgetting, ii) there may be fewer privacy-related issues in using generative replay techniques w.r.t. experience replay ones, iii) recalling past experiences through generation may be more akin on how biological systems work.

One of the earliest approaches is the Deep Generative Replay framework proposed in [160] in which a WGAN-GP [54] (the scholar model) is trained at time t by using both the currently available training data and the replay data generated using the generator obtained at time $t-1$. The performance of other generative models has been explored in [88] using mainstream benchmarks based on MNIST and CIFAR (mainstream benchmarks are discussed in detail in Chapter 2).

At the same time, it has been shown that the performance of algorithms that continually learn through generative replay drops when facing complex scenarios in which the training stream is composed of high-dimensional data [88, 175].

Structural techniques

Structural techniques try to tackle the forgetting problem by changing the architecture of the model. This usually includes adding learnable parameters, introducing masks, new layers, or even parallel copies of models/blocks.

Of the three macro-categories, this is the less explored one. This is also the most difficult category to subdivide as many approaches have been attempted. Here a tentative subdivision is attempted as *expansion techniques* and *masking techniques*.

Expansion techniques Expansion techniques are the most used structural techniques. These techniques aim at limiting the forgetting issue by allocating new learnable parameters (single units, layers, blocks, or even copies of the base model). These techniques usually start from known model architectures (possibly adapted to obtain a downsized form) or building blocks (such as ResNets blocks) and new parameters are added when deemed appropriate. These techniques usually try to circumvent the knowledge saturation problem that arises when the neural network has to learn a lot of different concepts.

Progressive Neural Networks (PNN) [147], which can be considered an extreme case of an expansion technique, is among the most known expansion technique. In PNN, a new copy of a base model is allocated for each new task to learn. Residual connections are inserted from previous to new models to allow for knowledge transfer and previous models are always frozen. Starting from this extreme technique, more recent attempts aimed at limiting the memory and computational overhead posed by PNN have been proposed [178, 91]. Instead of allocating whole parallel blocks, existing layers can be expanded or appended to the same branch, as proposed in [180]. An even more flexible capacity expansion technique is proposed in [188]. Both the last two techniques can also manage multi-task settings without forcefully splitting the model on a task basis. In fact, it must be noted that most of the structural techniques are designed for task-incremental scenarios (more in Section 2.1.1), and the additional learnable parameters allocated during past experiences are usually not allowed to change to allow for knowledge reuse.

Masking techniques This family includes all the techniques that re-use the existing model capacity by partitioning the model. These techniques, like the expansion ones, usually work in task-incremental settings only by creating a separate partition for each task. These techniques differ in how the training is carried out and how the partitions are created.

The model is usually partitioned by masking weights and/or activations on a per-task basis. At test time, the forward pass will apply these masks to compute outputs by using a partition of the model. The parts of the model allocated to a previously learned task can be frozen, which means that those techniques may be forgetting-free. Moreover, when learning new tasks, these techniques may reuse parameters and features used when learning previous tasks. Prominent strategies include HAT [152], PiggyBack [101], and Packnet [102]. Trajectory-based techniques such as the ones based on PathNets [43] are part of this sub-category, too.

One obvious negative aspect of these techniques is connected to the saturation of model capacity. However, this family of techniques may be used in conjunction with expansion techniques to prevent saturation issues.

1.3.2 Current challenges

Continual learning, even though early works can be dated back to 1989 [107], saw a surge in popularity only in recent years. Most of the efforts are directed towards the computer vision area, and in particular towards the supervised classification one. Because of that, many under-explored directions exist. With that, a set of challenges that must be tackled to reach the kind of intelligence one would like to achieve exist.

Adherence to reality and applications One of the biggest issues with continual learning research, which encompasses both techniques and benchmarks, is the lack of adherence to reality. Of course, the definition of *realistic* depends on the context. In general, many benchmarks don't seem to be able to model the kind of scenarios a real application would face (more on this in Chapter 2). Apart from that, strategies may need to evolve to accommodate for more complex scenarios, realistic images (or other types of data), realistic availability of labels and task signals, and to consider performance, memory, and time constraints.

We argue that, by adapting the benchmarks and the techniques used to tackle the forgetting phenomenon, one could enable practical applications to be implemented. Most of this thesis is centered around some proposed techniques and benchmarks that move in this direction.

Towards object detection Arguably, the majority of works found in the continual learning literature revolve around the object *classification* problem. However, this doesn't hold for the object detection problem. The importance of the classification problem cannot be argued, as it serves as the initial step towards building continuously learning systems for vision applications. However, detection could enable a more vast set of possible real-life applications and, in general, can be deemed a more realistic task for robotic applications.

The continual object detection area started to receive some attention only in recent times, and works in this direction [1] are still a few.

Instance vs category learning Most of the mainstream benchmarks found in the CL computer vision literature are based on well-known datasets used to evaluate the performance of models trained in a classic, offline, manner. Mainstream object datasets adopted in the CL literature include CIFAR10/100 and ImageNet. These datasets were designed to model the category classification model: the training set is usually composed of images of various objects (usually depicted only once throughout the whole dataset) and the goal is to learn how to label objects at the category level. On the contrary, the task of classifying at the *instance* level, in which the goal is to identify the specific object, is not as much explored. It must be noted that datasets and benchmarks such as the ones based on CORe50 [95] and OpenLORIS [158], which were designed specifically for benchmarking continual learning strategies, feature an instance-based classification problem. We argue that instance-based object classification could enable more adaptive robotic applications and *personal assistants*.

Streaming learning Incrementally training from experiences carrying very limited batches of data is a particularly interesting challenge as a strategy able to learn quickly from a very small set of instances could be used to enable continual real-time learning capabilities on *at-the-edge* devices, such as robots, embedded boards, and mobile systems.

Differently from the others, this research direction is a tad more explored [59, 99, 20]. Some works even focus on the extreme case in which the training dataset carried by each incremental experience is comprised of a single exemplar (*online streaming learning*).

Exploiting unlabeled data Unsupervised learning seems to be a natural element in any continually learning agent. An agent able to learn in an unsupervised way could autonomously refine its knowledge regarding the environment. Moreover, that could allow for some forward knowledge transfer, which is usually low (or none) in mainstream continual learning techniques. However, the most important benefit of using unsupervised learning techniques is the ability to use the vast amount of data obtained from sensors without the need for humans (or oracles) in the loop. That would reduce the gap between artificial and biological intelligent systems.

Recent works in the unsupervised learning field, such as the ones concerning the use of a contrastive loss [23], show that meaningful representations could be learned without any kind of supervision.

In addition, recent works in the continual learning field explicitly considered the unsupervised and semi-supervised setups [134, 2, 154, 179].

Chapter 2

Continual Learning Benchmarks

In this chapter, a comprehensive description of our efforts toward enabling continual learning in realistic and complex scenarios is given. The chapter starts with a description of the most popular benchmarks found in the continual learning literature. Each macro-category of benchmarks is analyzed to pinpoint their distinctive features as well as the issues which prevent them from modeling realistic situations.

Starting from these considerations, a set of novel benchmarks is proposed in Section 2.2. We believe that these benchmarks enable a more proper modelization of realistic situations for vision applications in which an agent has to learn incrementally while operating in an evolving environment. The techniques described in Chapter 3 are based on these premises.

2.1 Mainstream benchmarks and protocols

Before delving into the reasoning and details behind mainstream and the novel proposed benchmarks, it is better to spend a few words about elements such as common assumptions and the nomenclature on which continual learning benchmarks are based.

The ability of an agent to learn continually can be empirically assessed in different continual learning scenarios. Each scenario defines the constraints and the opportunities of the learning environment. In particular, it is assumed that continually learning agents should learn from a stream of data produced by nonstationary, dynamic environments [115, 87]. Since the data distribution may drift at any time, continual learning violates the i.i.d assumption behind traditional machine learning training procedures, giving rise to problems like catastrophic forgetting of previous knowledge [107].

The algorithmic engine enabling the continuous learning of new knowledge in intelligent agents is commonly named *strategy*. The issues faced when designing a continual learning strategy are heavily influenced by the specific implementation of the general continual learning scenario. The most popular scenarios all refer to an *experience*-based way of learning, in which the strategy should learn from novel data which becomes available over time.

In particular, in continual learning scenarios, learning is broken down into a, possibly unbounded, *stream of experiences* $S = e_1, e_2, e_3, \ldots$, with abrupt and instantaneous drifts between one experience and the other. Each experience e_i brings a set of data (*batch*), together with optional additional knowledge, like a *task label* (usually, a scalar value) which helps to uniquely identify the distribution generating the current data [87].

For the rest of this thesis, the terms *experience* and *batch* are often used interchangeably. It should be noted that the term *experience* is more general, as the term *batch* is already used in the Machine Learning literature with a slightly different meaning. Moreover, the term *batch* may be better suited to identify the training dataset carried by the experience, while the experience itself is not solely composed of the dataset it carries. For instance, experiences may carry additional signals such as task labels or boundaries. Finally, to tackle in advance a common cause of misunderstandings, it is common to find in literature the term *task* being used to refer to the element that is here called *experience*. The use of task labels is very popular in the continual learning literature and, as described in Section 2.1.1, in the popular Task-incremental scenario each experience carries all the training data connected to a single task. Because of this, the term *task* started to be used to identify an *experience*.

All the families of benchmarks described in the next sections start from some idea that is used to model a specific scenario. Each benchmark resulting from the application of that idea usually allows for the randomization of some of its defining elements. Running experiments multiple times on instantiations in which these parameters are randomly set allows for more robust results to be obtained. In this thesis, the term *run* is used to identify a single instantiation. For instance, in class-incremental benchmarks (described in Section 2.1.1), running experiments by randomly varying the order of encountered classes (that is, by experimenting on multiple runs) is a common practice used to stabilize the accuracy curve and to identify relevant variations (usually presented as the standard deviation).

Finally, the term *instance* is used to identify a single data point (for instance, an image in vision tasks), but the terms *exemplar* and *data sample* are widely used, too.

The scenarios described in the following sections all refer to classification problems (usually digits or object classification) where, for each exemplar, the target label is always given at training time. It must be noted that continual learning literature is not limited to the classification problem and fully-supervised learning schemes. However, efforts to explore these research directions are more recent in literature and are out of the scope of this thesis.

The following sections describe mainstream scenarios by following well-known subdivisions proposed in the literature [177, 115, 29]. However, for the sake of better framing the family of benchmarks that will be used as the reference for the rest of the thesis, the classification proposed in [95] and then refined in [103] is proposed as well. That subdivision is here used as a general way to contextualize scenarios based on their degree of *concept repetition*.

2.1.1 Class and Task Incremental

The surge of interest in continual learning has been initially driven by its application to deep learning methodologies and is mostly oriented towards supervised computer vision tasks, like object recognition from images [92, 147]. Naturally, one of the most intuitive procedures to convert available computer vision benchmarks into viable continual learning benchmarks is to concatenate multiple datasets to simulate drifts in the data distribution (one dataset per experience, as in the protocol used by

[92]). This allowed researchers to immediately leverage the vast amount of existing computer vision benchmarks and to rapidly test new continual learning strategies on large-scale streams. The learning objective was to classify patterns by assuming to know from which dataset each pattern arrived (referred to as a task-incremental learning scenario in the literature [177]). This task label information simplifies the continual learning problem since patterns from different datasets can be easily isolated by the model during both learning and inference.

After a period in which task-incremental has remained the most studied continual learning scenario, the attention of the community has now turned to class-incremental scenarios [135], where experiments are conducted on a single dataset, with patterns split by class and without any knowledge about the task label, neither during training nor during inference.

In the class-incremental setting, each experience e_i carries a training dataset $D_i = \{(x_j, y_j)\}_{j=1,\dots,M}$, where x_j is the input pattern and y_j is its target class. The peculiar characteristic of class-incremental scenarios is that they partition the target class space by assigning a disjoint set of classes to each experience. Formally, be $\mathcal{C} = \{c_k\}_{k=1,\dots,C}$ the set of all classes seen by the model during training and be $\mathcal{C}_i$ the subset of classes present in experience e_i, class-incremental scenarios satisfies the following condition:

$$\mathcal{C}_i \bigcap \mathcal{C}_j = \emptyset, \quad \forall i \neq j. \tag{2.1}$$

We will refer to the constraint expressed by Equation 2.1 as the *no repetition* constraint. It simply states that classes present in one experience are never seen again in future experiences or, likewise, that each class is present in one and only one experience.

The task-incremental scenario differs from the class-incremental one in which, in the task-incremental scenario, each class partition is treated as a separate task (subproblem) to be solved. In practice, this means that each training experience carries a different task label (usually an incremental scalar). This signal is used to identify the class partition both in the training and testing stages. That is, at test time, the task label is given for each test exemplar.

Following the subdivision proposed in [103], the scenarios based on the *no-repetition* condition can be gathered under the *New Classes (NC)* family of scenarios.

New Classes scenarios are nowadays very popular in the continual learning community. Their simplicity and ease of use greatly fostered new studies and efforts toward mitigating catastrophic forgetting, the main problem faced by models when learning in this setting.

2.1.2 Domain Incremental

While *New Classes* benchmarks feature a strict partitioning of classes encountered in the experience stream, the exact opposite happens in *New Instances (NI)* scenarios.

In these scenarios, experiences carry data belonging to the same data source and the task labels are not available (at least at test time). Depending on the specific incarnation, the classes that will be encountered may be or may be not already known from the beginning (that is, at least one instance for each class is available in experience "0"). Experiences may carry data from already seen classes (with no assumptions about the insertion of unexpected new classes), which enables for a rehearsal of already seen classes. In the mainstream literature, scenarios belonging (or akin) to the New Instances family have been instantiated in various ways and with different names, with Data and Domain-Incremental scenarios [177, 95, 103] being the most known.

The *New Instances* family of scenarios may fall into the exact opposite problem seen for *New Classes* scenarios: the *unbounded repetition* of concepts. It must be noted that this idea is more aligned with a realistic scenario, in which the same classes may be encountered many times, as it happens in the life of human beings. This positive element has been kept into consideration when designing the benchmark proposed later in Section 2.2.

2.1.3 Issues with mainstream benchmarks

The aforementioned families of scenarios are based on a few assumptions that make it difficult to use them to model a realistic scenario of continual learning for vision problems.

Experience composition Let us consider some of the limitations caused by the *no repetition* constraint of New Classes scenarios. In many real-world environments, repetition occurs naturally. This means that class/task-incremental learning is not able to model scenarios in which repetition comes directly from the environment. Examples include robotic manipulation of multiple objects, prediction of financial trends, autonomous driving tasks, etc. A learning agent exposed to a continuous stream of information should be able to incrementally acquire new knowledge, but also forget unnecessary concepts and prioritize learning based on some notion of importance. Not all the perceived information should be treated equally: if a certain pattern never occurs again, it may be useless to still pretend to predict it correctly. The statistical re-occurrence of concepts and their temporal relationship could be considered important sources of information to help determine the importance of what the agent perceives [104, 27]. It is very hard to discern what to forget and what concepts to reinforce if all the information is treated equally.

Learning in a compartmentalized fashion hinders many of the possible insights an agent may draw from the complexity of the environment, eventually limiting its possibility to create its world model suitable to the changing task it has to tackle.

Another important side effect of the *no repetition* constraint is that the lack of repetition induces large forgetting effects. However, this forgetting is artificially induced. Focusing on catastrophic forgetting would not be inconvenient if real-world problems were aligned with the characteristics of the class-incremental scenario [169]. However, this is not the case as repetition occurs naturally during the lifetime of biological agents. The issues raised by the usage of New Classes scenarios can be easily addressed by relaxing the *no repetition* constraint.

The New Instances scenarios are more able to model how human beings explore the environment during their lifetime as the same concept is experienced more than once on different occasions. This means that the *no repetition* constraint is not an issue in this scenario. However, as already discussed in Section 2.1.2, depending on the amount of variation in the experiences, the data-incremental scenario may not present a large amount of forgetting, as it may collapse in an i.i.d. stream of experiences featuring an unrealistic *unbounded amount of repetitions*. This means that

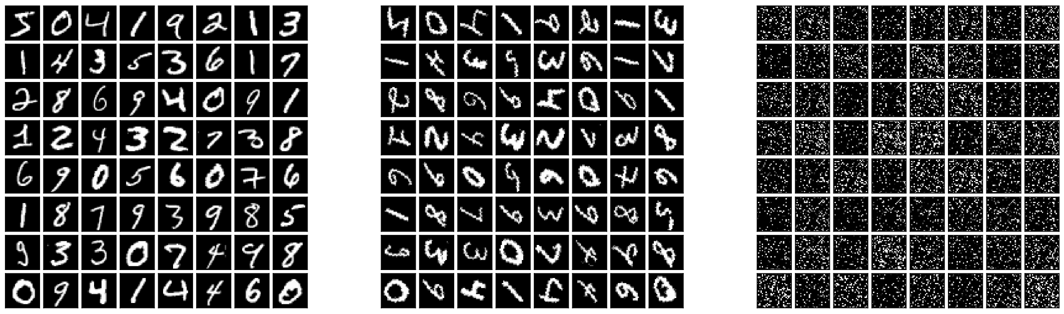

FIGURE 2.1: MNIST (left) and derived Rotated-MNIST (middle) and Permuted-MNIST (right) are among the most used datasets in continual learning literature.

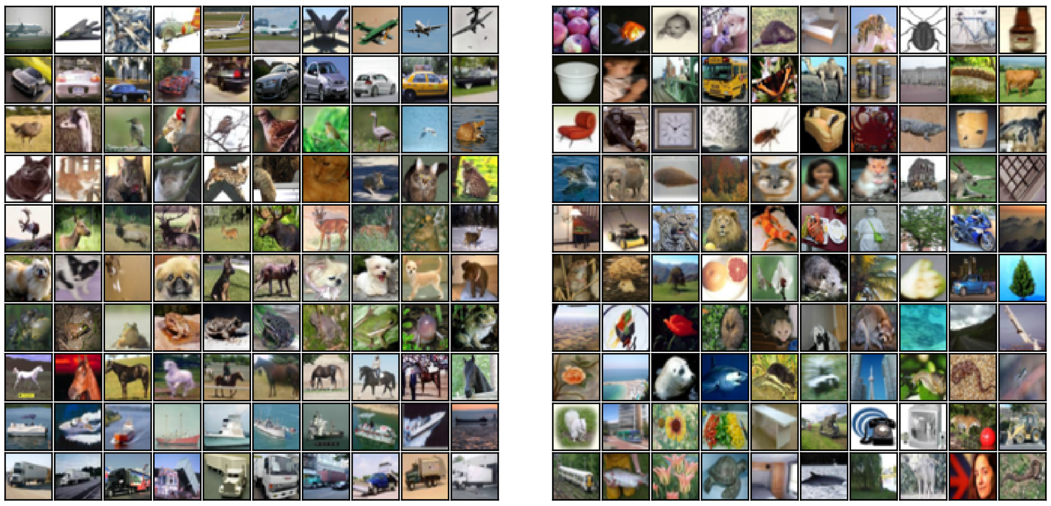

FIGURE 2.2: CIFAR-10 (left) and CIFAR-100 (right) are among the most used object classification datasets in continual learning literature.

these scenarios are an excellent starting point, but some additional constraints need to be added to produce more realistic benchmarks.

Realism of content Another issue with mainstream scenarios in the vision area is the very content of the used datasets. For historical reasons, the early works in the continual learning field started with simple benchmarks. The idea was to show the relevance of the catastrophic forgetting phenomenon [107] and then, later in time, to show how the first proposed strategies could defy forgetting when working with simple, low dimensional, input images. Because of this, MNIST [86] (Figure 2.1) and CIFAR [81] have arguably been the most popular datasets employed in continual learning literature so far.

These datasets share two issues that prevent them from being a proper base for realistic benchmarks for robotic vision applications: *low dimensionality* and *unrealistic content*. The first issue is self-evident: MNIST features single-channel images at a 28x28 resolution[1] while CIFAR10 and CIFAR100 feature 32x32 RGB images[2]. On the content side, MNIST cannot be taken as a good base for benchmarking strategies that

[1]MNIST website: http://yann.lecun.com/exdb/mnist/.

[2]CIFAR website: https://www.cs.toronto.edu/~kriz/cifar.html.

have to deal with natural images, but it remains a good choice when first prototyping an idea for a continual learning strategy. Concerning CIFAR, its images depict real objects, but their low resolution prevents important details concerning both the object itself and the surrounding environment to emerge. In natural images, details are very important: edges, color and luminosity patches, distracting elements in the background (etcetera) all may deeply impact the complexity of the task at hand.

Sparse images vs video streams Apart from the use of natural images, another element of interest for vision applications is the choice of format in which the visual stream is presented to the system. Most of the benchmarks make use of datasets composed of sparse images, in which instances are not correlated with each other. On the contrary, biological systems can learn from streams of images captured in real-time. The use of datasets composed of video sessions instead of sparse images seems more reasonable as this would reduce the gap between the experimental setup and the real scenario.

2.2 The New Instances and Classes scenario

The *New Instances and Classes (NIC)* scenario was conceived as a more realistic benchmark family in which the assumptions found in the other two families of benchmarks are not present. That is, differently from NI and NC, NIC was conceived to not fall in the *no-repetition* and *unbounded repetition* extreme cases. In *New Instances and Classes* scenarios, new training patterns belonging to both known and new classes become available in subsequent training batches. This has no counterpart among other benchmarks discussed so far. In fact, to the best of our knowledge, almost no study explicitly addresses the NIC scenario, which we deem as the most natural setting for many applications such as robotics vision, where: i) a variable number of small non-i.i.d. training batches are encountered over time; ii) training batches may contain objects already seen before as well as completely new objects.

In contrast, benchmarks of the *New Instances* family introduce a strong priori regarding the availability in time of instances for already seen classes. That is, the assumption that experiences will carry non-i.i.d batches of data is relaxed, thus allowing for an implicit rehearsal.

Benchmarks from the *New Classes* family introduce the idea of encountering all instances of a set of classes only once (in a single experience). This has two immediate consequences. i) A specific class timeline is introduced. When the end of the training stream is reached, classes seen at the beginning of the benchmark will usually be less recognized, while classes seen towards the end of the experiences stream will usually feature a higher recognition ratio. This is why, for New Classes benchmarks, the average accuracy over time is usually a preferred metric in place of the final accuracy. ii) At the same time, the base assumption makes it possible to specifically craft the algorithmic solution to make the most of the base assumption. This becomes even more relevant when task labels are assumed to be available at both training and test time because this allows for a simplified subdivision of the problem. On the contrary, when designing a solution for a benchmark from the New Instances and Classes family, one cannot count on the aforementioned assumptions. And, in general, continual learning algorithms designed to tackle New Classes benchmarks can hardly be applied to NIC (or even NI) scenarios.

NIC benchmarks can be seen as a more generalized class of benchmarks. However, with this increased generality, several new elements are to be considered. How should be experiences crafted? Should they contain instances from a limited set of classes? Should they contain (more or less) the same amount of instances? At which time should be novel classes introduced? Should experience "0" mimic the availability of richer a set of base classes? In addition, how can one generate such benchmarks from a given dataset? Most of the benchmarks used in the continual learning literature are built upon existing datasets by applying some algorithm able to generate corresponding instantiations (which usually differ from each other due to the randomization of some elements). This is why "New Instances and Classes (NIC)" should be considered a *family* of benchmarks: many benchmarks can be created by changing the number of experiences, their content, etcetera.

For instance, some researchers pointed out that reducing the size of training batches makes continual learning more challenging [103, 58, 135]. However, one may wonder what is the lower bound for the size of training batches and if it is feasible to train a system by gradient descent with very small non-i.i.d. incremental batches each containing few images of a single class. It is well known that stochastic gradient descent (SGD) works well with large and i.i.d. mini-batches, but this assumption is difficult to meet in realistic scenarios. Let us consider a robot that is learning to recognize some objects shown by an operator (one at a time). In an ideal application, when a new object is shown, the robot acquires a short video and immediately updates its knowledge to become able to recognize the new object. The frames extracted from the video would constitute one or more small mini-batches containing highly correlated patterns from a single class: a rather challenging setting to face with standard SGD-based optimization techniques. At the same time, introducing constraints such as i) the order in which new classes are to be inserted, or ii) if the same class can be seen again, or iii) how often a previous class should be rehearsed, may be unrealistic and may hinder the robot usability. It follows that this scenario can't be easily modeled by the means of a New Classes or New Instances benchmark. At the same time, this scenario can be modeled by the means of a NIC benchmark. However, properly setting the aforementioned elements (amount of experiences, content, etc) is necessary to obtain realistic benchmarks.

2.2.1 New Instances and Classes v2 (NICv2)

A NIC protocol was initially introduced in [95]. That NIC protocol was specifically tailored for the CORe50 dataset and only one specific instantiation of it was proposed: NIC-79.

In that benchmark, the first training batch contains 10 classes (∼3,000 images) and each of the subsequent 78 incremental experiences includes about 1,500 images of 5 classes. However, as shown in Figure 2.4 (left), the random generation procedure used in [95] produces a sequence where almost all the classes are introduced in the first 10-15 batches, making that protocol very close to a NI scenario. Figure 2.4 (left) clearly shows this issue, making it clear that for the vast majority of the scenario no new classes are encountered.

To make the benchmark more challenging and closer to a real application where new objects can be discovered also later in time, we proposed a new three-way protocol named NICv2.

Algorithm 1 NICv2 generation pseudocode: the first batch includes one training session of a single class within each category. Then for each of the remaining 40 classes, we randomly sample the minimum allowed insertion point and then assign all the remaining training sessions to batches in the permitted range. The size of training batches (checked at line 14) depends on num_batches (see Table 1). Note that the algorithm might loop for unappropriated values of some parameters.

```
1: procedure NICv2(num_runs, num_batches, max_start)
2:     num_runs: number of sequences to produce. Since in continual learning the pattern presentation order has an impact on the accuracy, experiments need to be averaged over multiple runs. In this paper we used $num\_runs = 10$.
3:     num_batches: the total number of training batches (refer to Table 1).
4:     max_start: we need to limit the maximum position for the insertion point of classes to leave some room to accommodate all their training sessions.
5:     for each run in num_runs:
6:         assign to $B_1$ 10 training sessions (by selecting 1 class from each category)
7:         for each class $C$ of the ramaining 40 classes:
8:             random sample $insertion\_point \in [1, max\_start]$
9:             for each training sessions $S$ of class $C$:
10:                $assigned = false$
11:                while not assigned:
12:                    random sample current batch $B_c$ with $c \in [insertion\_point, num\_batches]$
13:                    if $B_c$ is not full:
14:                        assign training session $S$ to $B_c$
15:                        $assigned = true$
```

The NICv2 protocol was conceived with the idea of balancing the class first introduction over the training batches. This is achieved by forcing the class first introduction to be evenly distributed across the batches thus producing *runs* that are both more challenging and realistic (see Figure 2.4, right). In general, the NICv2 protocol is able to recreate realistic scenarios in which no general assumption can be made on the order and distribution in time in which new classes may be introduced.

In addition, differently from the original NIC protocol, NICv2 can be applied to different datasets. Critical elements such as the number of experiences and the moment in which new classes are introduced can be customized. The pseudo-code used to generate the benchmarks using the NICv2 protocol is reported in Algorithm 1.

2.2.2 NICv2 for CORe50

CORe50 [95] was specifically designed as an object recognition video benchmark for continual learning. It consists of 164,866 128$\times$128 images of 50 domestic objects belonging to 10 categories (see Figure 2.3); for each object, the dataset includes 11 video sessions ($\sim$300 frames recorded with a Kinect 2 at 20 fps) characterized by relevant variations in terms of lighting, background, pose and occlusions. Video sessions feature an egocentric vision of hand-held objects. This allows for emulating a scenario where a robot has to incrementally learn to recognize objects while manipulating them. Objects are presented to the robot by a human operator who can also provide the labels, thus enabling a supervised classification. Such an applicative scenario is well described in [118, 119, 117]. Because of these peculiar features, we deemed CORe50 to be a perfect dataset to be used when benchmarking the continual learning techniques described later in this chapter.

For instance, let us consider a scenario in which an embodied agent is operating in a complex environment subject to frequent and unpredictable changes. To operate

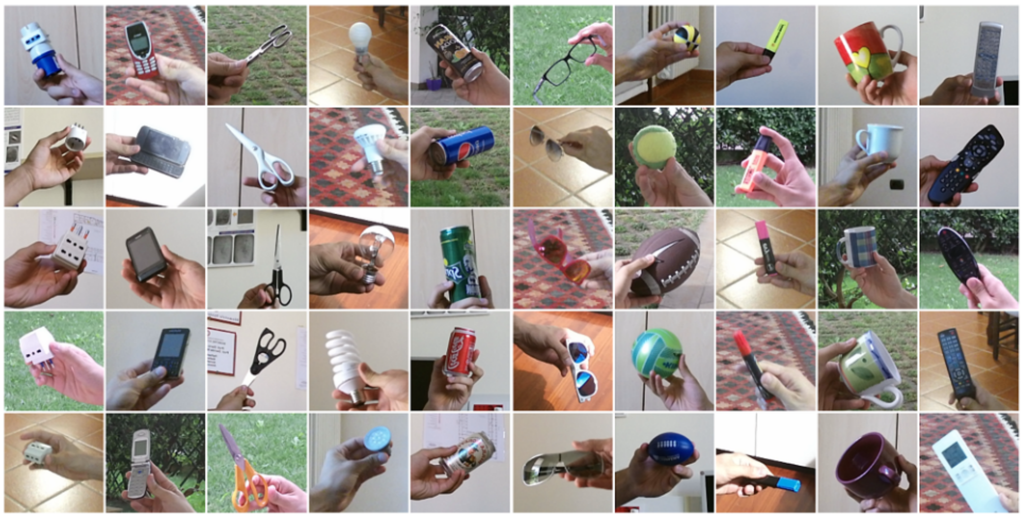

FIGURE 2.3: Example images of the 50 objects in CORe50, the continual learning video dataset used in this paper. Each column denotes one of the 10 categories. Classification experiments in this paper are object-based, so each object corresponds to a class.

TABLE 2.1: Batch number and composition in NIC and NICv2.

Protocol	# batches	Initial batch		Incremental Batches	
		# Classes	# Images	# Classes	# Images
NIC	79	10	3,000	5	1,500
NICv2-79	79	10	3,000	5	1,500
NICv2-196	196	10	3,000	2	600
NICv2-391	391	10	3,000	1	300

in this ever-changing environment, the agent is required to learn and adapt continuously. For example, in the context of object recognition, a robot should be able to learn (without forgetting) objects belonging to never seen classes as well as improve its recognition capabilities as new instances of already known classes are discovered. Ideally, the continual learning process should be triggered by the availability of short videos of single objects and performed online on onboard hardware with fine-grained updates.

To model this rather complex scenario, we applied the NICv2 generation technique to CORe50 by progressively reducing the size of the batch of data carried by each experience. This, in turn, led to a higher number of fine-grained experiences. In particular, in CORe50-NICv2-391 (referred to as just "NICv2-391" from now on), each of the 390 incremental experiences includes only one training video session (∼300 images) of a single class (see Figure 2.5). It follows that this benchmark is a well-suited modelization of the aforementioned scenario. At the same time, the NICv2 generation procedure allows for the modeling of less extreme scenarios: we also propose NICv2-79, which fixes the issues found with the original NIC protocol (see Figure 2.4), and NICv2-196 (see Table 2.1).

For CORe50, the test set used for NICv2 is the default test set shared by all the CORe50 protocols [95]; it includes 3 sessions for each class, with a null intersection

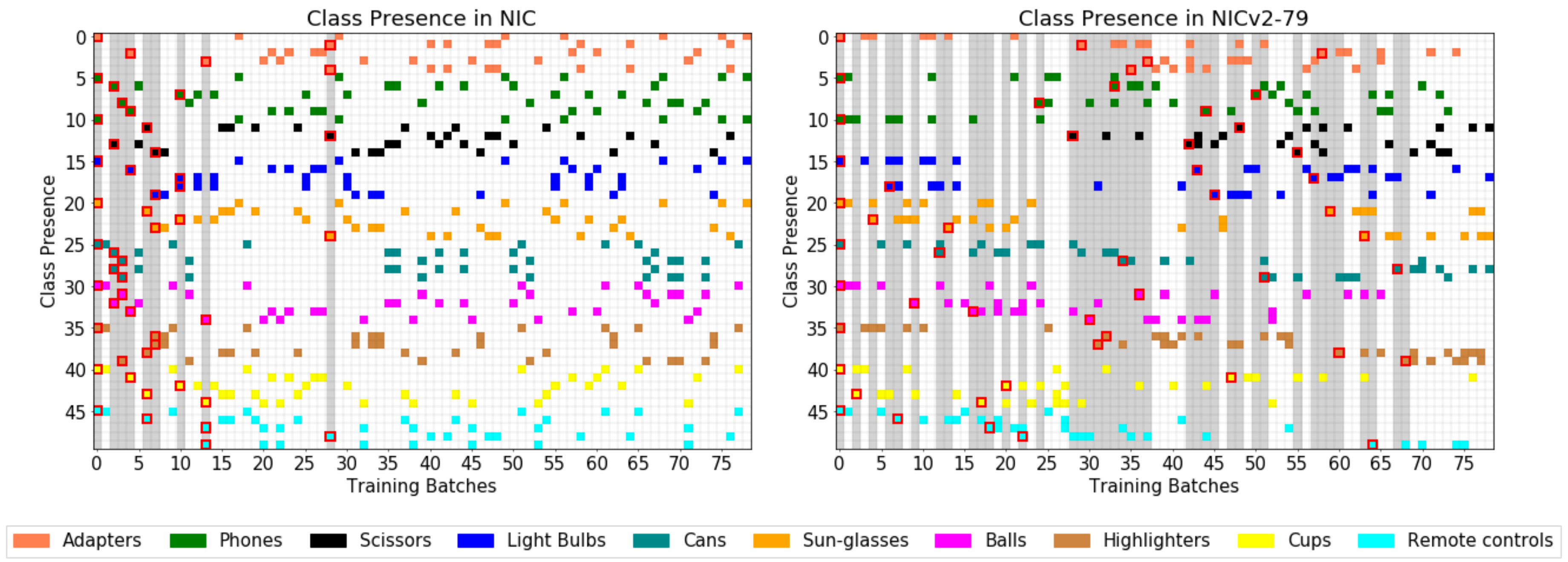

FIGURE 2.4: Classes encountered over time in the first run of NIC (left) and NICv2-79 (right). Each row denotes a class; colors are used to group the 50 classes (objects) of the CORe50 dataset in its object 10 categories. Each column denotes a training experience. A colored block in a (row, column) cell is used to indicate that at least one training video session of the row class is present in the column batch. Each row contains a maximum of 8 colored blocks because each class has 8 training sessions in the training set (the remaining 3 sessions are left out for the test set). The red-framed cells denote the first introduction of a class. Gray vertical bands highlight experiences where at least one class is seen for the first time.

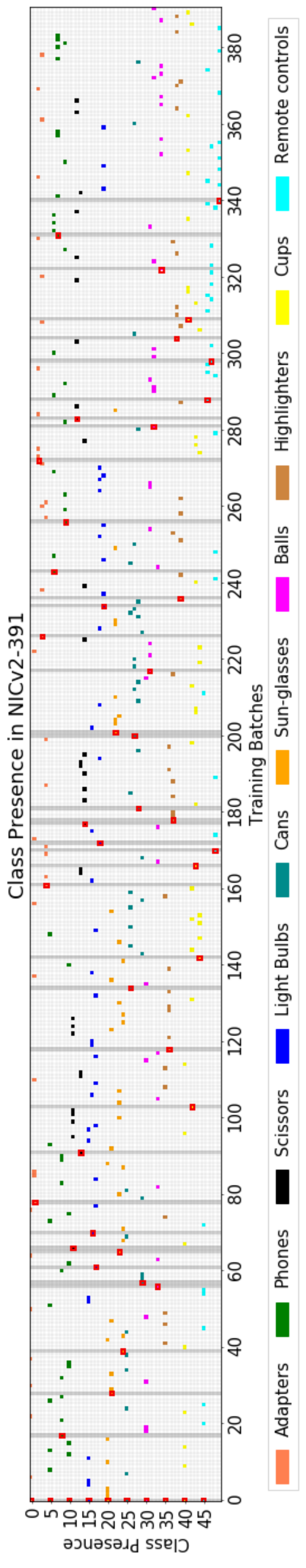

FIGURE 2.5: Classes encountered over time in the first run of NICv2-391. Each row denotes a class; colors are used to group the 50 classes into the 10 categories. Each column denotes a training experience. A colored block in a (row, column) cell is used to indicate that at least one training session of the row class is present in the column batch. Differently from the NICv2-79 and 196 scenarios, in NICv2-391 we forced each experience to contain only instances from one class (by allocating a single video session to each experience). Class presentation ordering for the first run of NICv2-196 is reported in Figure A.2 of the appendix.

with training batches. To speed up the evaluation process (which requires one evaluation after each training batch) we sub-sampled the test set by selecting one frame every second (from the original 20 fps). Because of the high correlation among successive frames in the sequences, such a strong sub-sampling is not reducing the test set variability and the accuracy results on the original and the downsampled version are very close. We made available at `https://vlomonaco.github.io/core50` all the file lists of the new NICv2 protocols along with the down-sampled test set.

2.2.3 NICv2 for other datasets

As already mentioned, one of the strengths of NICv2 is its flexibility which makes it possible to apply it to other datasets (other than CORe50).

The base assumption on which NICv2 is built is that the reference training dataset can be split into *semantic unit*s. NICv2 allocates each of these units to a single experience, and each experience can contain one or more semantic units. In CORe50, a semantic unit is a single video session (which depicts a single object). In general, each semantic unit should describe a batch of correlated data (in which the correlation is given by some semantic).

As such, NICv2 can be naturally applied to any video-based dataset and it is not limited to CORe50 only. For instance, we successfully applied NICv2 to a novel video dataset we gathered for evaluating the performance of the CORe Android application (more on this in Section 5.1). However, NICv2 could be applied to other kinds of datasets as well. If no semantic subdivision is possible, the dataset content could be arbitrarily split into fixed-size experiences (for instance, using a random ordering and splitting).

Chapter 3

Continual Learning Algorithms in Complex Scenarios

In the previous chapter, mainstream benchmarks have been described along with their strengths and issues (Section 2.1). Following that, the NICv2 benchmarks were introduced (Section 2.2), which serve as the reference benchmark for the efforts described in this chapter. These benchmarks try to model a realistic stream of experiences in which novel classes may be inserted at unpredictable moments in time. In addition, the NICv2-391 benchmark based on the CORe50 dataset [95] features a strongly non-i.i.d. composition of experiences.

The goal of the techniques proposed in this chapter is to allow for efficient and effective continual learning on constrained devices such as robots, mobile, and embedded systems. These techniques were designed to seamlessly handle the insertion of new classes at random moments, to handle non-i.i.d experiences, while at the same time keeping their resource (computation, memory, time) usage low. The findings described in this chapter have been used to design and develop real applications, such as the ones described in Chapter 5.

3.1 AR1* and CWR*: replay-free approaches

A simple solution to deal with a rather long stream of experiences would be to store all the data and cyclically re-train the entire model from scratch [76]. However, this approach is rather impractical when learning from high-dimensional streaming data, especially in highly constrained computational platforms and embedded systems [87, 165].

Some replay-based techniques have been proposed to mitigate the forgetting problem, which becomes difficult to handle when learning from a long stream of experiences: by maintaining some representative patterns from past experiences, new frames can be interleaved with past ones in each mini-batch. However, this involves extra memory (to store the past data) and computation (due to a higher number of forward/backward steps): when designing AR1* and CWR* we asked ourselves whether continual learning over small non-i.i.d. batches *is feasible without replay*.

This section introduces two novel replay-free continual learning techniques, CWR* and AR1*, that can learn effectively even when applied to the challenging NICv2-391 benchmark (described in Sec 2.2.2). In particular, experiments show that AR1* can outperform other state-of-the-art replay-free techniques by more than 15% accuracy in some cases, with a very light and constant computational and memory overhead across training batches.

While this section explains how the difficult NICv2-79, -196, and -391 benchmarks can be tackled without replaying instances from past experiences, in Section 3.2 a novel technique named Latent Replay is described, which can be easily integrated into AR1*, CWR*, and other popular algorithms (such as LwF) as well.

3.1.1 Evolving AR1 and CWR

In [103] it was shown that a simple approach like CWR+, where the fully connected layer is implemented as a double memory, is quite effective to control forgetting in the NC scenario. However, after the first training batch, CWR+ freezes all the layers except the last one, thus losing the benefit of an incremental adaptation of the underlying representation. AR1 [103] was then proposed to extend CWR+ by enabling end-to-end continual training throughout the entire network; for this purpose, the Synaptic Intelligence [189] regularization approach (similar to EWC [79]) is adopted to constrain the change of critical weights.

In the following subsections we:

1. show how we adapted CWR+ to tackle NIC scenarios, thus making it able to reload past weights for already known classes and to adapt them with weighted contributions from different batches. As AR1 incorporates CWR+ in its main algorithm, this modification will result in the CWR* and AR1* continual learning strategies (Section 3.1.1).
2. show that, in a complex scenario with small and non-i.i.d. batches, Batch Normalization layers thwart the continual learning process, and replacing them with Batch Renormalization [70] can effectively tackle this problem (Section 3.1.1).
3. propose a selective weight freeze for the CNN models adopting Depth-Wise Separable Convolutions. We show that altering spatial filters has a strong impact in terms of forgetting (Section 3.1.1).
4. reduce the computational and storage complexity of AR1 (and in general of EWC-like approaches), by introducing an alternative way to implement weights update starting from the Fisher matrix (Section 3.1.1).

While 1. is specific to CWR+, 2., 3. and 4. can be applied to several other CL approaches as well.

From CWR+ to CWR*

CWR+, whose pseudo-code is reported in Algorithm 2 of [103] and in Algorithm 5 in the appendix, maintains two sets of weights for the output classification layer: cw are the consolidated weights used for inference and tw the temporary weights used for training; cw are initialized to 0 before the first batch and then iteratively updated, while tw are reset to 0 before each training batch.

In Algorithm 2, we propose an extension of CWR+ called CWR* which works under both NC and NIC scenarios; in particular, under NIC the coming batches include patterns of both new and already encountered classes. For already known classes, instead of resetting weights to 0, we reload the consolidated weights (see line 7).

Furthermore, in the consolidation step (line 13) a weighted sum is now used: the first term represents the weight of the past and the second term is the contribution from the current training batch. The weight $wpast_j$ used for the first term is proportional to the ratio $\frac{past_j}{cur_j}$, where $past_j$ is the total number of patterns of class j encountered in past batches whereas cur_j is their count in the current batch. In case of a large number of small non-i.i.d. training batches, the weight for the most recent batches may be too low thus hindering the learning process. In order to avoid this, a square root is used in order to smooth the final value of $wpast_j$.

Algorithm 2 CWR* pseudocode: $\bar{\Theta}$ are the class-shared parameters of the representation layers; the notation $cw[j]$ / $tw[j]$ is used to denote the groups of consolidated / temporary weights corresponding to class j. Note that this version continues to work under NC, which is seen here a special case of NIC; in fact, since in NC the classes in the current batch were never encountered before, the step at line 7 loads 0 values for classes in B_i because cw_j were initialized to 0 and in the consolidation step (line 13) $wpast_j$ values are always 0.

```
1: procedure CWR*
2:     $cw = 0$
3:     $past = 0$
4:     init $\bar{\Theta}$ random or from pre-trained model (e.g. on ImageNet)
5:     for each training batch $B_i$:
6:       expand output layer with neurons for the new classes in $B_i$ never seen before
7:       $tw[j] = \begin{cases} cw[j], & \text{if class } j \text{ in } B_i \\ 0, & \text{otherwise} \end{cases}$
8:       train the model with SGD on the $s_i$ classes of $B_i$:
9:         if $B_i = B_1$ learn both $\bar{\Theta}$ and $tw$
10:        else learn $tw$ while keeping $\bar{\Theta}$ fixed
11:      for each class $j$ in $B_i$:
12:        $wpast_j = \sqrt{\frac{past_j}{cur_j}}$, where $cur_j$ is the number of patterns of class $j$ in $B_i$
13:        $cw[j] = \frac{cw[j] \cdot wpast_j + (tw[j] - avg(tw))}{wpast_j + 1}$
14:        $past_j = past_j + cur_j$
15:      test the model by using $\bar{\Theta}$ and $cw$
```

Replacing Batch Normalization with Batch Renormalization

Batch Normalization (BN) [71] is widely used in modern deep neural networks to control internal covariate shift, thus making learning faster and more robust. In BN the mini-batch moments (i.e., mean μ_{mb} and variance σ^2_{mb}) are used to normalize the input values x_i as:

$$\hat{x}_i = \frac{x_i - \mu_{mb}}{\sqrt{\sigma^2_{mb} + \epsilon}} \tag{3.1}$$

where ϵ is a small constant added for numerical stability, and the normalization is per-channel. However, if mini-batches are small and/or non-i.i.d., the mini-batch moments are not stable and BN can fail. A natural solution to reduce the moment fluctuations would be replacing μ_{mb}, σ^2_{mb} with global values μ, σ computed as moving averages over an initial (large-enough) training batch. After all, this is the standard approach when switching from training to inference. However, as argued in [71], using moving averages to perform the normalization during training does not produce the desired effects since gradient optimization and the normalization counteract each other, possibly leading the model to diverge.

Batch Renormalization (BRN) was proposed in [70] to deal with small and non-i.i.d. mini-batches. In BRN the normalization takes place as follows:

$$\hat{x}_i = \frac{x_i - \mu_{mb}}{\sigma_{mb}} \cdot r + d \tag{3.2}$$

$$r = \frac{\sigma_{mb}}{\sigma}, \quad d = \frac{\mu_{mb} - \mu}{\sigma} \tag{3.3}$$

where μ, σ are computed as moving averages during training. By expanding r and d in the equation 3.2, we obtain $\hat{x}_i = \frac{x_i - \mu}{\sigma}$ which clearly highlights the dependency on the global moments. A further step is suggested in [70] to clip r in $[\frac{1}{r_{max}}, r_{max}]$ and d in $[-d_{max}, d_{max}]$. It is worth noting that when $r = 1$ and $d = 0$, then BRN≡BN; hence, by properly setting r_{max} and d_{max} the behavior of BRN can be moved from a pure BN to a more stable normalization based on global statistics. In practice, the author of [70] recommend to perform an initial stage by keeping $r_{max} = 1$, $d_{max} = 0$ in order to stabilize the moving averages μ, σ and then progressively increasing r_{max} and d_{max} to 3 and 5, respectively.

Continual learning over small batches is an emblematic case of small and non-i.i.d. minibatches. For example, in NICv2-391 (described in Section 2.2.2) each training batch includes 300 patterns from a single class, and even using a mini-batch size of 300 (the full batch) patterns remain strongly correlated. Our first attempts to learn continuously over a so long sequence of one-class batches were unsatisfactory. Even for the most accurate strategies (e.g., AR1*) accuracy slightly increased in the first batches from 13% to 15% but then remained steady and lower than 16-17%. We initially thought that the reason was the single-class mini-batches, making the problem a sort of one-class classification with no negative examples. However, upon replacement of BN with BRN and a proper parametrization, we were able to continuously learn over small batches with unexpected efficacy (see Section 3.1.2).

Depthwise Layer Freezing

Depth-Wise Separable Convolutions (DWSC) are quite popular nowadays in many successful CNN architectures such as MobileNet [66, 149], Xception [26], EfficientNet [166]. Classical filters in CNN are shaped as 3D volumes. For example, a 5×5×32 filter spans a spatial neighborhood of 5×5 along 32 feature maps; on the contrary, in DWSC we first perform 32 5×5×1 spatial convolutions (an independent convolution on each feature maps) and then combine results with a 1×1×32 filter working as a feature map pooler. Advantages in terms of computation and weight reduction have been pointed out by several researchers.

Inspired by previous findings with Hierarchical Temporal Memories [137, 104] where gradient descent by HSR only affects coincidence pooling, here we propose to fine-tune DWSC architectures by freezing depthwise spatial filters and leaving pointwise poolers free to learn. We speculate that modifying a spatial filter (i.e. the way a local neighborhood is processed) can be detrimental in terms of forgetting during continual learning because it alters the semantics of what upper layers have already learned; on the other hand, feature map pooling, which can be seen as a way to promote feature invariance, is less prone to concept drifts.

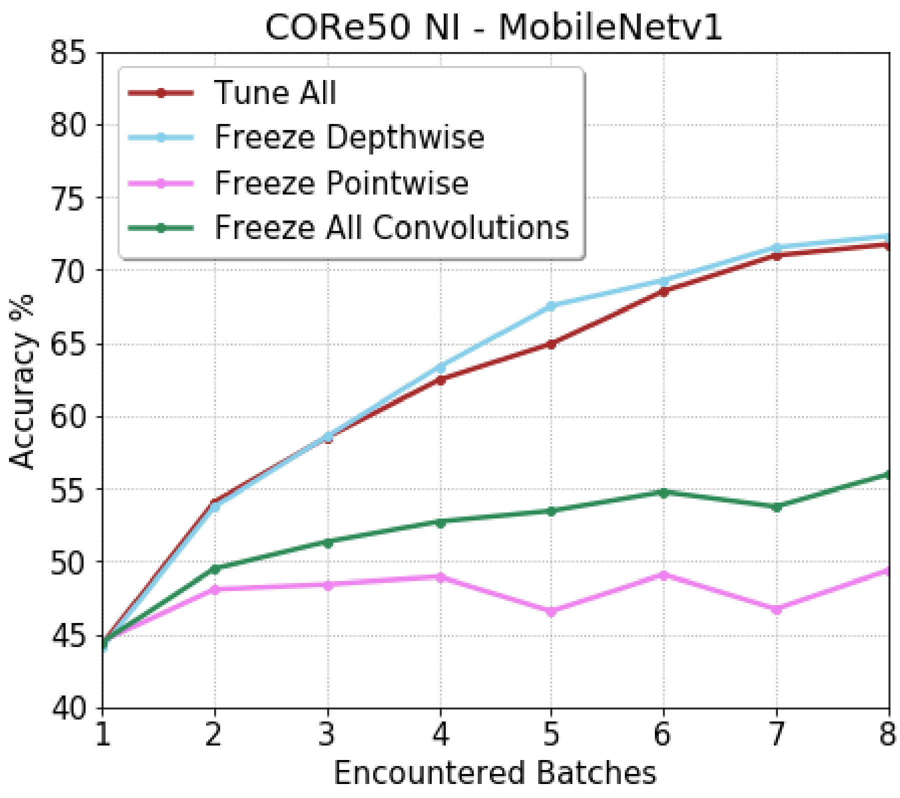

FIGURE 3.1: Continual Learning on CORe50, SIT - NI scenario. In the NI scenario, all the 50 classes are discovered in the first batch and successive training batches provide new instances of the already known classes; however, since past instances are not retained, the incremental process is prone to forgetting. In this experiment, a MobileNet (pre-trained on ImageNet) is incrementally tuned (naive strategy) over 8 batches with different weight freeze strategies. Each curve is averaged over 10 runs where the batch order is randomly permuted.

A simple experiment is illustrated in figure 3.1, where a MobileNet is incrementally fine-tuned on the 8 learning batches of CORe50, SIT - NI scenario [95]. Here, no specific measure is put in place to control forgetting except early stopping the gradient descent after 1 epoch (naive strategy). The four curves denote the classification accuracy when: i) all the weights are tuned; ii) weights of depthwise convolution layers are frozen; iii) weights of pointwise convolution layers are frozen; iv) weights of all convolution layers are frozen. Note that weights of fully connected layers (e.g. output layer) are never frozen. The proposed strategy (case ii) achieves the best result and, with respect to a full tuning, allows skipping some gradient computations and can reduce the amount of memory used to store weight-associated data[1]. The complementary strategy (case iii) is the worst one, thus confirming that altering spatial filters has a strong impact in terms of forgetting.

[1]In per-weight adaptive learning rate methods (such as Adam [78]) extra values (i.e. running averages) need to be stored for each "free" weight. Further, if a regularization method based on Fisher matrix is used (such as EWC [79]) we need to store the optimal value for previous tasks and the Fisher value for each weight.

Weight Constraining by Learning Rate Modulation

The Elastic Weight Consolidation (EWC) approach [79] tries to control forgetting by selectively constraining the model weights which are deemed to be important for the previous tasks. To this purpose, in a classification approach, a regularization term is added to the conventional cross-entropy loss, where the weights θ_k of the model are pulled back to their optimal value θ_k^* with strength F_k proportional to their importance for the past:

$$L = L_{cross}(\cdot) + \frac{\lambda}{2} \cdot \sum_k F_k \cdot (\theta_k - \theta_k^*)^2. \tag{3.4}$$

Synaptic Intelligence (SI) [189] is a lightweight variant of EWC where, instead of updating the Fisher matrix F at the end of each batch[2], F_k are obtained by integrating the loss over the weight trajectories exploiting information already available during gradient descent. For both approaches, the weight update rule corresponding to equation 3.4 is:

$$\theta_k^{'} = \theta_k - \eta \cdot \frac{\partial L_{cross}(\cdot)}{\partial \theta_k} - \eta \cdot F_k \cdot (\theta_k - \theta_k^*) \tag{3.5}$$

where η is the learning rate. This equation has two drawbacks. Firstly, the value of λ must be carefully calibrated: if its value is too high, the optimal value of some parameters may be overshot, leading to divergence (see discussion in Section 2 of [103]). Secondly, two copies of all model weights must be maintained to store both θ_k and θ_k^*, leading to double memory consumption for each weight. To overcome the above problems, we propose to replace the update rule of equation 3.5 with:

$$\theta_k^{'} = \theta_k - \eta \cdot (1 - \frac{F_k}{max_F}) \cdot \frac{\partial L_{cross}(\cdot)}{\partial \theta_k} \tag{3.6}$$

where max_F is the maximum value for weight importance (we clip to max_F the F_k values larger than max_F). Basically, the learning rate is reduced to 0 (i.e., complete freezing) for weights of highest importance ($F_k = max_F$) and maintained to η for weights whose $F_k = 0$. It is worth noting that these two updated rules work differently: the former still moves weights with high F_k in the direction opposite to the gradient and then makes a step in direction of the past (optimal) values; the latter tends to completely freeze weights with high F_k. However, in our experiments with AR1, the two approaches lead to similar results, and therefore the second one is preferable since it solves the aforementioned drawbacks.

3.1.2 Experimental Results

We run several experiments on CORe50 NICv2, to validate the approaches introduced in Section 3.1.1 and to compare them with a naive baseline and three state-of-the-art replay-free approaches. In particular, for all the experiments, the following techniques have been considered:

- CWR*: the extension of CWR+ introduced in Section 3.1.1.

[2]In this paper, for the EWC and AR1 implementations we use a single Fisher matrix updated over time, following the approach described in [103].

- AR1*: the approach introduced in [103], here implemented by replacing CWR+ with CWR* and by adopting the weight constraining by learning rate modulation introduced in Section 3.1.1.
- Naive: a baseline technique where we simply continue gradient descent along the training batches and the only measure to control forgetting is early stopping.
- EWC and LFW: the techniques originally introduced in [79] and [92] and adapted to continual learning in SIT scenario as detailed in [103].
- DSLDA: the strategy recently proposed in [59], where an on-line extension of the Linear Discriminant Analysis (LDA) classifier [133] is trained on top of a fixed deep learning feature extractor. DSLDA obtained state-of-the-art accuracy on CORe50 (10 categories setting) [59], even outperforming replay-based techniques such as iCaRL [135] and ExStream [58].
- Cumulative: the upper bound, in which the model is trained on the union of the current batch and all the past data.

For all the experiments we used a MobileNet v1 [66] with: $width\ multiplier = 1$, $resolution\ multiplier = 0.5$ (input 128×128), pre-trained on ImageNet. MobileNet architectures provide a good tradeoff in terms of accuracy/efficiency and, in our opinion, are well suited for porting continual learning at the edge.

For all the above techniques the MobileNet v1 architecture was modified by replacing the 27 Batch Normalization layers with corresponding Batch Renormalization layers and using (for training) a mini-batch size of 128 patterns. We used Batch Renormalization implementation for Caffe [73] made available in [13]. This modification improves the accuracy of all the methods, making CWR* and AR1* able to learn also in the case of 391 single class batches. Batch Renormalization hyperparameters and their schedule have been experimentally set as follows:

- **Batch 1**: for the first 48 iterations we keep $r_{max} = 1$, $d_{max} = 0$ to startup the global moments; then, we progressively move r_{max} to 3 and d_{max} to 5 (as suggested in [70]). The weight of the past when updating the moving averages was set to 0.99, as suggested for $(1 - \alpha)$ in [70].
- **Subsequent batches**: global moments computed on batch 1 are inherited by batch 2 and slowly updated across the batch sequence. In this case, we noted that continual learning over small non-i.i.d. batches benefits from more stable moments, and therefore the weight of the past for updating moving averages was set to 0.9999. Here we have no startup phase for the global moment so the values of r_{max} and d_{max} are kept fixed across all the iterations of the batches. While using the suggested values of $r_{max} = 3$ and $d_{max} = 5$ still works, we noted that reducing them (i.e. relaxing batch renormalization constraints) brings some befits. More details about the experiments and the hyperparameters used are provided in the appendix.

For all the techniques we also applied depthwise layer freezing (as introduced in Section 3.1.1) starting from Batch 2. This can be simply implemented by setting the learning rate to 0 for the 14 non-pointwise convolution layers (13 depthwise + 1 3D) in MobileNet v1 architecture. While in NICv2 experiments this had a negligible impact on the accuracy, we found it can be advantageous in other scenarios (see NI curves in Figure 3.1) and, in general, this reduces computations/storage during

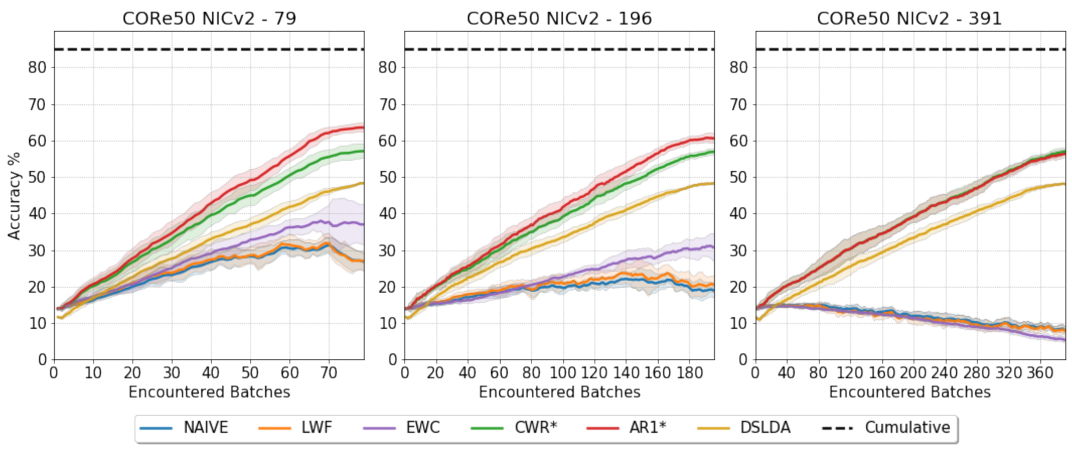

FIGURE 3.2: Continual learning accuracy along the incremental training experiences over NICv2-79, NICv2-196, and NICv2-391. None of the methods compared uses replay techniques. Naive refers to a simple approach where the model is tuned along the experiences and the only protection against forgetting is early stopping. LWF and EWC are well-known methods for CL (see [92, 79]). DSLDA is a recently proposed streaming continual learning approach [59]. The black dashed line denotes the "upper bound" accuracy achieved by the Cumulative approach (a full training on the entire dataset). Each experiment was averaged on 10 runs. Colored areas represent the standard deviation of each curve[3].

SGD (fewer gradient calculations, lower memory to accumulate per weight extra-data, etc.).

Figure 3.2 shows the results of our experiments on NICv2-79, NICv2-196, and NICv2-391. The curves are averaged over 10 runs where the training batch order is randomly shuffled. Hyperparameters of the methods have been coarsely tuned (i.e., without any systematic grid search) on run 0 and then kept fixed for the other 9 runs.

It can be noted that CWR* and AR1* show a very good learning trend across training batches, with only a minor drop in accuracy when the batch granularity decreases. The accuracy near linearly increases for most of the batches and slows down in the final part of the sequences; we believe this is not caused by the saturation of learning capabilities but is more likely due to the absence of examples of new classes in the final part of the sequences (see Figure 2.2.1b). Standard deviation across runs is also quite small denoting good stability. Naive, LWF, and EWC exhibit fair performance on 79 batches but their efficacy significantly decreases with 196 batches and are not able to learn in the most challenging case of 391 single-class batches. DSLDA accuracy is quite good and stable but remains lower than CWR* and AR1* in all three settings. The advantage of AR1* over CWR* (due to the extra freedom to improve the representation) reduces as the batch size decreases and is null for 391 batches. We speculate that, in this case, the gradient steps induced by small and highly non-i.i.d. mini-batches tend to overfit the mini-batches themselves with no improvement in terms of generalization.

Figure 3.3 compares AR1* accuracy in the configuration with Batch Normalization and Batch Renormalization. It is evident that for 391 batches Batch Normalization heavily hurt the learning capabilities. However, it is worth noting that Batch Renormalization brings some advantages to continual learning even when using larger batches that may include patterns from more than one class.

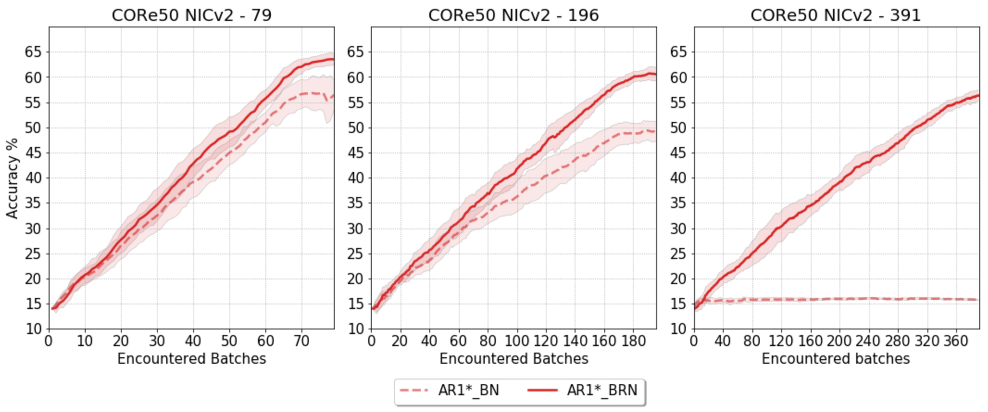

FIGURE 3.3: Comparing AR1* accuracy results in the Batch Normalization vs Batch Renormalization configurations on the three NICv2 benchmarks.

3.1.3 Performance analysis

In order to better understand and compare the performance of the proposed continual learning strategies, in Table 3.1 we also report the total run time, the maximum external memory size (where patterns from previous batches are stored), and the number of additional trainable parameters introduced while learning across the NICv2-391 batches. All the metrics are averaged across 10 runs.

		Memory Overhead	
Strategy	Run Time (m)	Data (MB)	Params (MB)
CWR*	21,4	0	0,2
Naive	25,6	0	0
LWF	27,8	0	0
EWC	31,2	0	24,4
AR1*	**39,9**	**0**	**12,2**
DSLDA	79,1	0	0,2
Cumul.	2826,2	4712,3	0

TABLE 3.1: Total run time (in minutes, for both training and test), memory overhead (in terms of maximum data storage for replay and number of additional trainable parameters introduced) for each strategy on the NICv2-391 protocol. Please note that: *(i)* all the replay-free strategies hereby listed have a *constant* memory / computational overhead which is fixed and independent from the number of training batches processed; *(ii)* the Cumulative metrics are computed considering a re-training from scratch after each incremental batch.

Replay-free approaches show a remarkable advantage w.r.t. the cumulative upper bound (where the model is re-trained from scratch after each incremental batch on the cumulated data), both in terms of speed-up and in terms of total memory overhead. Among them, AR1* shows the best trade-off between accuracy and efficiency with about 40 minutes to complete the run and a fixed memory overhead of only 12.4 MB for handling the additional parameters of the learning rate modulation introduced in Section 3.1.1. We would also underline that the current Synaptic Intelligence implementation embedded in AR1* is not optimized (gradient is recomputed in python outside the Caffe framework) without exploiting the data already available from SGD. We believe that upon proper optimization, AR1* efficiency can be very close to Naive one.

Finally, it is worth noting that the advantage of weight constraining by learning rate modulation (introduced in Section 3.1.1) for AR1* is negligible in terms of accuracy (less than 0.1% average improvement in NICv2-79) but relevant in terms of per weight storage since we do not need to store about 3,2 million θ_k^* values.

3.1.4 Remarks on AR1* and CWR*

The aforementioned results show that replay-free continual learning techniques can learn over long sequences of small and highly correlated batches, even in the challenging case of one class at a time. In fact, CWR* and AR1* showed a good (near-linear) learning trend across the training batches and proved to be very robust even with small one-class batches. On the other hand, well-known CL techniques such as EWC and LWF are not able to learn effectively in our experiments. We speculate that: (i) a regularization technique alone is not effective to protect important weights in the upper levels when dealing with a large number of small batches; (ii) learning the upper layer(s) "in isolation", as CWR* and AR1* do, is very important for continual learning, especially in SIT setting. DSLDA, which recently achieved state-of-art accuracy on some continual learning benchmarks, performed quite well in our experiments, but its accuracy and efficiency are lower than CWR* and AR1*.

Of course, when approaching such complex scenarios, other continual learning techniques should be considered. For example, here we did not compare AR1* and CWR* with replay-based approaches such as iCaRL [135] and GEM [99] because, even if using an external memory to store past data may simplify the task, it brings drawbacks in terms of extra storage/computations. Actually, some preliminary comparisons of CWR+, AR1, and DSLDA with replay-based approaches had already been reported in [103] and [59] for CORe50 (NC scenario) showing that the proposed replay-free approaches are still competitive when a moderate number of patterns is maintained in the external memory by iCaRL and GEM (2,500-4,500 training images). Another interesting technique, reporting good results on CORe50, is the Dual-Memory Recurrent Self-Organization proposed in [116]: however, the results included in that work are not directly comparable with our achievements because the aforementioned approach also exploits the temporal dimension of CORe50 videos (by using temporal windows instead of single frames).

3.2 Latent Replay: an efficient replay mechanism

Training on the edge (e.g., on light computing devices such as smartphones, smart cameras, embedded systems, and robotic platforms) is highly desirable in several applications where privacy, lack of network connection, and fast adaptation are real constraints. While some steps in this direction have been recently moved [90], training on the edge often remains unfeasible. In fact, given the high demand in terms of memory and computation, most machine learning models nowadays are trained on powerful multi-GPU servers, and only frozen models are deployed to edge devices for inference.

Furthermore, in some applications (e.g., robotic vision, see Figure 3.10), training a deep model from scratch as soon as new data becomes available is prohibitive in terms of storage/computation even if performed server-side. Continual learning techniques, where complex models are incrementally trained on small batches of

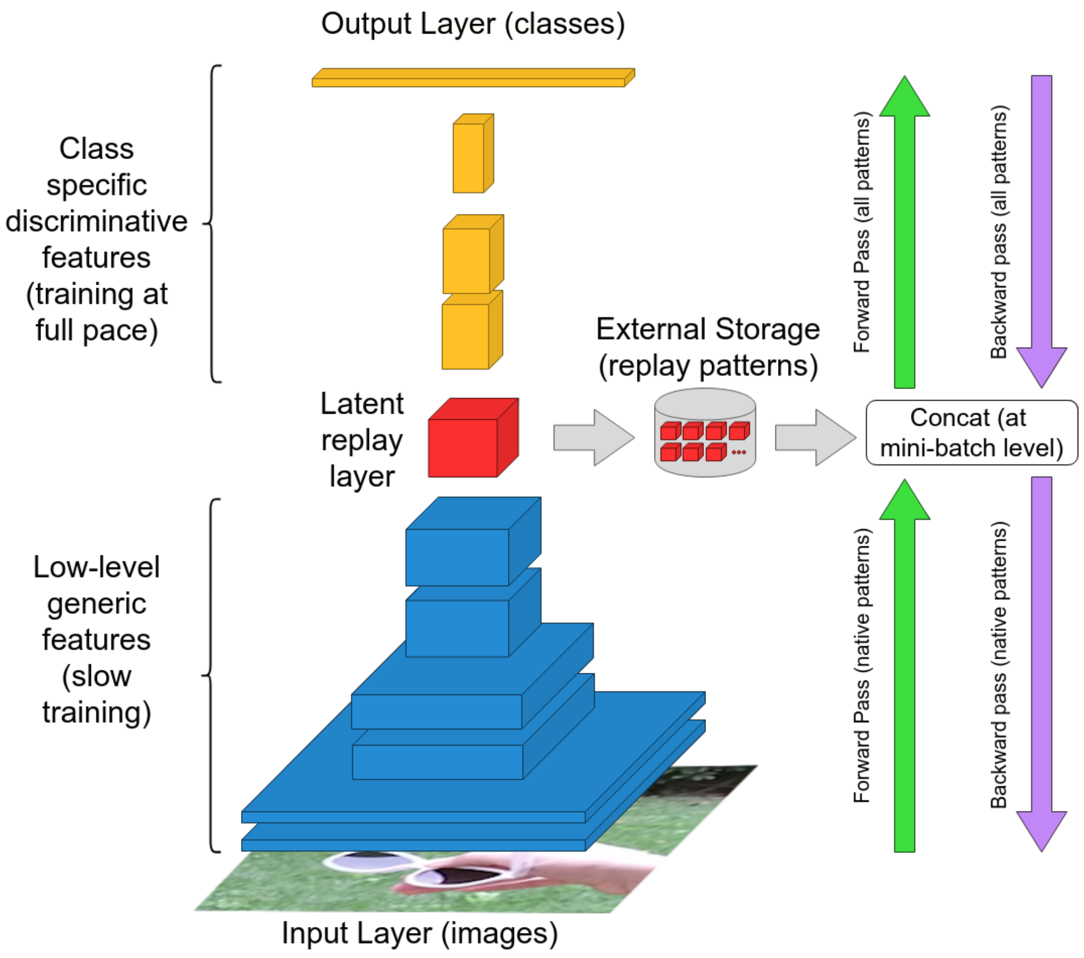
Output Layer (classes)
Class specific discriminative features (training at full pace)
Latent replay layer
External Storage (replay patterns)
Concat (at mini-batch level)
Forward Pass (all patterns)
Backward pass (all patterns)
Forward Pass (native patterns)
Backward pass (native patterns)
Low-level generic features (slow training)
Input Layer (images)

FIGURE 3.4: Architectural diagram of Latent Replay.

new data, can make the learning problem tractable even for CPU-only embedded devices enabling remarkable levels of adaptiveness and autonomy.

AR1* and CWR*, which are replay-free, showed good results in tackling the complex NICv2 benchmarks without exploiting replay techniques (see Section 3.1.2). While those results are encouraging, the top accuracy reached by AR1* at the end of the training sequence is in the range of 55-65% depending on the batch granularity, and the gap w.r.t. cumulative training (∼85%) exploiting all the data at one time is quite relevant (>20%, see Figure 3.2). Patterns replay proved to be an effective approach to contrast forgetting in continual learning scenarios [135, 99, 143, 125]. Periodically replaying some representative patterns from old data helps the model retain important information from past tasks/classes while learning new concepts. iCaRL [135] uses well-designed entry/exit criteria (denoted as herding) to maintain a class-balanced set of exemplars that maximize representativeness. For this purpose, we show that a small amount of pattern replay is sufficient to significantly improve accuracy on the NICv2-391 benchmarks (Section 3.2.1). For completeness, another class of replay techniques named Generative Replay (also known as *"Pseudo-rehearsal"* [142]) exists, where surrogates of past data are generated without explicitly storing raw instances. These techniques look very appealing because of the storage saving; however, most of the proposed approaches to date do not allow online generation of effective replay patterns.

No matter which specific replay technique is employed, replay mechanisms introduce both memory (storage) and computational overhead. For standard replay techniques, the memory occupation depends on the dimensionality and on the number of instances to be stored in the replay buffer, which may be an issue when handling high dimensional data streams. On the computation side, the constant refresh significantly increases the required amount of computational resources because of the extra forward and backward steps and this makes the resulting training too resource-demanding for real-time applications. In general, replay is difficult to apply on edge devices, where both storage and computational resources may be limited.

Starting from the results of AR1* and CWR* described in Section 3.1, and considering the preliminary results obtained by applying a simple replay mechanism, we designed the Latent Replay technique. Different from AR1* and CWR*, Latent Replay is a more general technique that can be used to handle replay in conjunction with existing continual learning algorithms.

When designing the Latent Replay technique, our goal was to *reduce as much as possible the gap w.r.t. the cumulative upper bound* and, at the same time, to *provide an efficient implementation strategy of CL approaches to enable nearly real-time training on the edge*.

In Section 3.2.3 we compare our approach with several continual learning algorithms and show its advantages in reaching state-of-the-art performances on two different benchmarks: CORe50 (NICv2 benchmarks) and OpenLORIS. In addition, we show that latent activations can be sparsified and compressed with almost no accuracy reduction, thus allowing for a reduction of the space used to store the replay buffer.

Apart from the theoretical and experimental results here described, this technique was successfully used to enable continual learning on smartphone devices (Section 5.1), and to enable continual learning on ultra-low-power embedded boards (Section 5.2). In addition, we used Latent Replay in conjunction with LwF [92] in the continual learning competition held at IROS 2019 (more info in Sections 3.2.3 and 5.4.1), obtaining 2nd place in the Object Classification track.

3.2.1 Native Replay

In [103] it was shown that a very simple replay implementation (hereafter denoted as *native rehearsal* or *native replay*), where for every training batch a random subset of the batch patterns is added to the external storage to replace a (equally random) subset of the external memory, is not less effective than more sophisticated approaches such as iCaRL. Therefore, we opted for simplicity and started by expanding CWR* and AR1* with the trivial replay approach summarized in Algorithm 3. In Figure 3.5 we compare the learning trend of CWR* and AR1* of a MobileNetV1[4] trained with and without replay on CORe50 NICv2-391. We use the same protocol and hyper-parameters introduced in Section 3.1 and a replay buffer of 1500 patterns. It is well evident that even a moderate external memory (about 1.27% of the total training set) is very effective to improve the accuracy of both approaches and to reduce the gap with the cumulative upper bound that for this model is ∼85%.

To understand the influence of the external memory size we repeated the experiment with different RM_{size} values: 500, 1000, 1500, and 3000. Since replay itself protects the model from forgetting, we also run AR1* (where important weights of lower

[4]The network was pre-trained on ImageNet-1k.

Algorithm 3 Pseudocode explaining how the external memory RM is populated across the training batches. Note that the amount h of patterns to add progressively decreases to maintain a nearly balanced contribution from the different training batches, but no constraints are enforced to achieve a class-balancing.

```
1: $RM = \varnothing$
2: $RM_{size}$ = number of patterns to be stored in $RM$
3: **for each** training batch $B_i$:
4:     train the model on shuffled $B_i \cup RM$
5:     $h = \frac{RM_{size}}{i}$
6:     $R_{add}$ = random sampling $h$ patterns from $B_i$
7:     $R_{replace} = \begin{cases} \varnothing & \text{if } i == 1 \\ \text{random sample } h \text{ patterns from } RM & \text{otherwise} \end{cases}$
8:     $RM = (RM - R_{replace}) \cup R_{add}$
```

layers are protected from forgetting by using Synaptic Intelligence [189] regularization) without Synaptic Intelligence protection, that is lower layers weights are left totally unconstrained; in the following, we denote this approach as AR1*free. The results are shown in Figure 3.6: it is worth noting that increasing the replay memory leads to better accuracy for all the algorithms, but the gap between 1500 and 3000 is not large and we believe 1500 is a good trade-off for this dataset. AR1*free works slightly better than AR1* when a sufficient number of replay patterns are provided but, as expected, accuracy is worse with light (i.e. $RM_{size} = 500$) or no replay.

It is worth noting that the best combination in Figure 3.6 (AR1*free with 3000 patterns) is only 5% worse than the cumulative upper bound and a better parametrization and exploitation of the replay memory could further reduce this gap.

3.2.2 Latent Replay

In deep neural networks the layers close to the input (often denoted as representation layers) usually perform low-level feature extraction and, after a proper pre-training on a large dataset (e.g., ImageNet), their weights are quite stable and reusable across applications. On the other hand, higher layers tend to extract class-specific discriminant features and their tuning is often important to maximize accuracy.

With *latent replay* (see Figure 3.4) we denote an approach where, instead of maintaining copies of input patterns in the external memory in the form of raw data, we store the activations at a given layer (denoted as *Latent Replay layer*). To keep the representation stable and the stored activations valid we propose to slow down the learning at all the layers below the latent replay one and to leave the layers above free to learn at full pace. In the limit case where lower layers are completely frozen (i.e., slow down to 0) latent replay is functionally equivalent to replaying from the input, but achieves a computational and storage saving thanks to the smaller fraction of patterns that need to flow forward and backward across the entire network and the typical information compression that networks perform at higher layers.

In the general case where the representation layers are not completely frozen, the activations stored in the external memory suffer from an aging effect (i.e., as the time passes they tend to increasingly deviate from the activations that the same pattern would produce if feed-forwarded from the input layer). However, if the training of these layers is sufficiently slow, the aging effect is not disruptive since the external

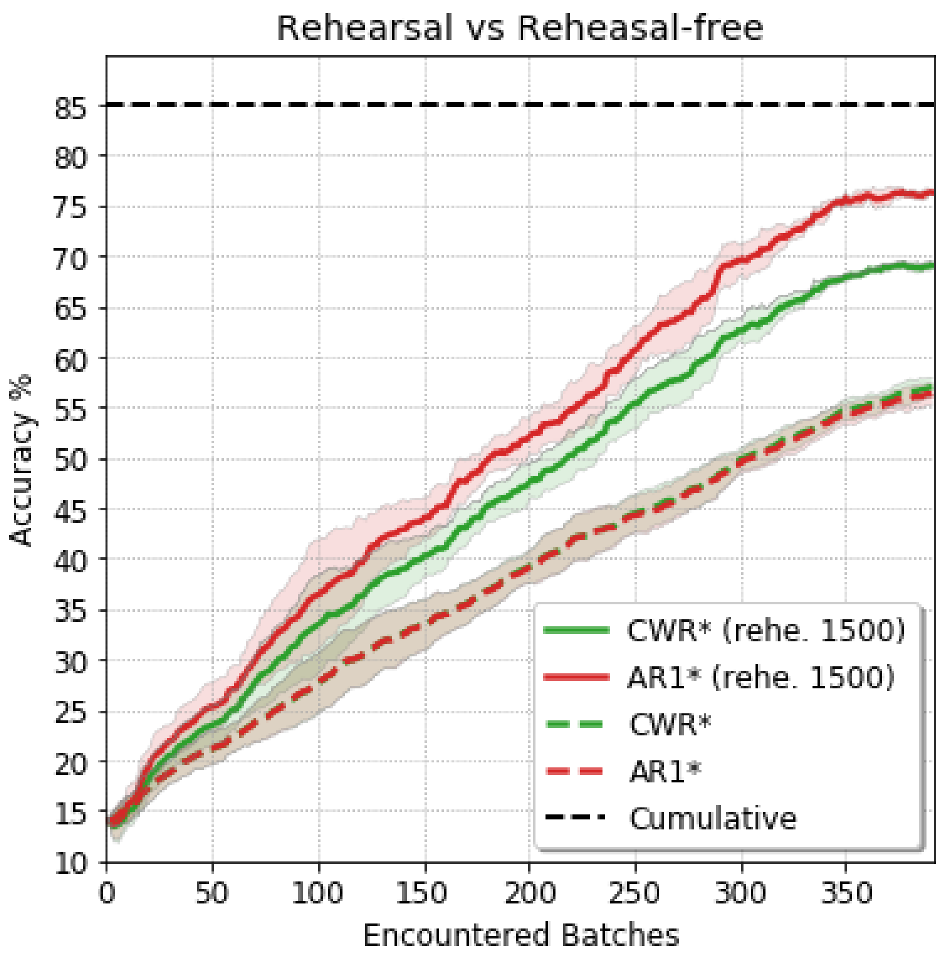

FIGURE 3.5: Comparison of CWR* and AR1* on CORe50 NICv2-391 with and without rehearsal ($RM_{size} = 1500$). Each experiment was averaged on 5 runs with different batch ordering: colored areas represent the standard deviation of each curve. The black dashed line denotes the reference accuracy of the cumulative upper bound.

memory has enough time to be rejuvenated with fresh patterns. When latent replay is implemented with mini-batch SGD training: *(i)* in the forward step, a concatenation is performed at the replay layer (on the mini-batch dimension) to join patterns coming from the input layer with activations coming from the external storage; *(ii)* the backward step is stopped just before the replay layer for the replay patterns.

3.2.3 Experiments and Results

Hereafter, while the proposed latent replay approach is architecture agnostic, we discuss its specific design with several continual learning algorithms CWR*, AR1*, AR1*free, and LWF over a MobileNet [66] pre-trained on ImageNet-1K. We compare our latent replay approach with other state-of-the-art techniques on benchmarks based on CORe50 [95] and OpenLORIS [158].

Experiments on CORe50 NICv2-391

For the CORe50 experiments:

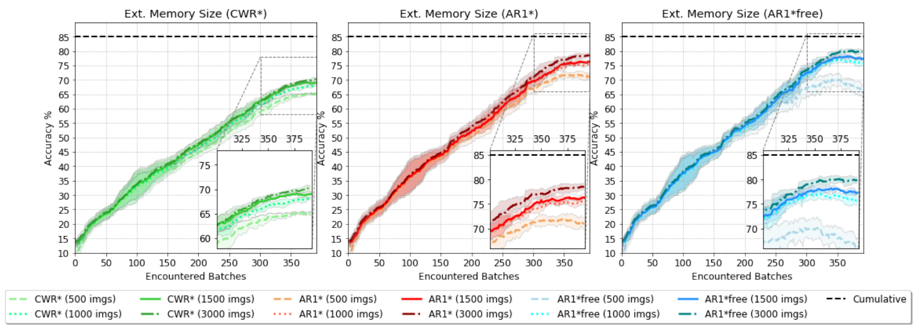

FIGURE 3.6: Comparison of CWR*, AR1*, and AR1*free on CORe50 NICv2-391 with different external memory sizes ($RM_{size} = 500, 1000, 1500$ and 3000 patterns).

- we use a MobileNetV1 and focus on CWR* and AR1*, which have been already proved to be competitive on this benchmark;
- for all the methods, the output layer (`fc7`) must be implemented as a double memory with proper (pre)initialization and (post)fusion for each training batch (for details see Section 3.1.1);
- for CWR*, the latent replay layer is the second-last layer (i.e., `pool6`);
- for AR1* and AR1*free, the latent replay layer can be pushed down and selected according to the accuracy-efficiency trade-off discussed below;
- for AR1*free, the Synaptic Intelligence regularization is switched off.

To simplify the network design and training, we keep the proportion of original and replay instances fixed: for example, if the training batches contain 300 instances and the external memory 1500 instances, in a mini-batch of size 128 we concatenate 21 ($128 \times 300/1800$) original instances (of the current experience) with 107 ($128 \times 1500/1800$) replay instances. In this case, only 21 instances (over 128) need to travel across the blue layers in Figure 3.4.

Concerning the learning slow-down in the representation layers, we found that an effective (and efficient) strategy is blocking the weight changes after the first batch (i.e., learning rate set to 0) while leaving the batch normalization moments free to adapt to the statistics of the input instances across all the experiences. Following the findings described in Section 3.1.1, in which it was shown that replacing BN with Batch Renormalization (BRN) [70] is a very important element needed to achieve handle fine-grained non-i.i.d. experiences, we replaced BN layers with BRN layers. In the context of latent replay, if we leave the BRN moments free to adapt, the activations stored in the external memory suffer the aging effect described in Section 3.2.2. However, we experimentally verified that, by proper setting of the global moment mobile windows (more details are provided in Appendix B.2), the accuracy drop due to the aging effect is quite limited and in any case the final accuracy is higher w.r.t. the case where BRN moments in the representation layers are frozen. On the computational side, blocking the weight changes in the representation layers allows skipping the backward pass in the lower part of the network also for native patterns, since updating the BRN moments only relies on the forward pass.

In Figure 3.7, we report the accuracy of AR1*free with latent replay ($RM_{size} = 1500$) for different choices of the replay layer (reported between parenthesis). As expected,

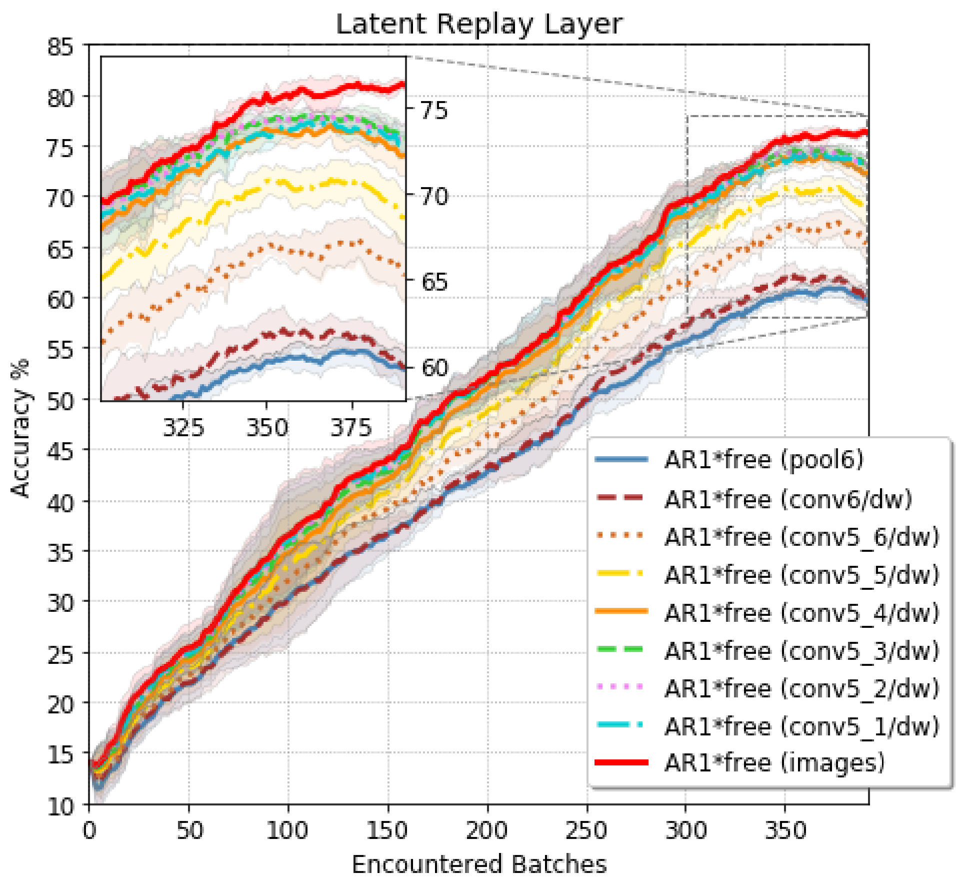

FIGURE 3.7: AR1*free with latent replay ($RM_{size} = 1500$) for different choices of the latent replay layer. Setting the replay layer at the `pool6` layer makes AR1*free equivalent to CWR*. Setting the replay layer at the "images" layer corresponds to native replay (same curve of Figure 3.6 for AR1*free and 1500 instances). The saturation effect which characterizes the last training batches is due to the data distribution in NICv2-391 (see [96]): in particular, the lack of new instances for some classes (whose data was already introduced) slows down the accuracy trend and intensifies the effect of activations aging.

when the replay layer is pushed down the corresponding accuracy increases, proving that a continual tuning of the representation layers is important. However, after `conv5_4/dw` there is a sort of saturation and the model accuracy is no longer improving. The residual gap (∼4%) with respect to *native rehearsal* is not due to the weight freezing of the lower part of the network, but to the aging effect introduced above. This can be simply proved by implementing an "intermediate" approach that always feeds the replay pattern from the input and stops the backward at `conv5_4`: such an intermediate approach achieved an accuracy at the end of the training very close to the native rehearsal. We believe that the accuracy drop due to the aging effect can be further reduced by better tuning of BNR hyper-parameters and/or with the introduction of a scheduling policy that can make the global moment mobile windows wider as the continual learning progresses (i.e., more plasticity in the early stages and more stability later).

On the Computation, Storage and Accuracy Trade-off To better evaluate the latent replay w.r.t. native rehearsal, in Table 3.2 we report the relevant dimensions:

(i) computation refers to the percentage cost in terms of operations of a partial forward (from the latent replay layer on) relative to a full forward step from the input layer; *(ii)* input size is the dimensionality of the instance to be stored in the external memory (considering that we are using a MobileNetV1 with $128 \times 128 \times 3$ inputs to match CORe50 image size); *(iii)* accuracy and Δ accuracy quantify the absolute accuracy at the end of the training and the gap with respect to a native rehearsal, respectively.

Layer	Computation % vs Native Rehearsal	Instance Size	Final Accuracy	Δ Accuracy % vs Native Rehearsal
Images	100.00%	49152	77.30%	0.00%
conv5_1/dw	59.261%	32768	72.82%	-4.49%
conv5_2/dw	50.101%	32768	73.21%	-4.10%
conv5_3/dw	40.941%	32768	73.22%	-4.09%
conv5_4/dw	**31.781%**	**32768**	**72.24%**	**-5.07%**
conv5_5/dw	22.621%	32768	68.59%	-8.71%
conv5_6/dw	13.592%	8192	65.24%	-12.06%
conv6/dw	9.012%	16384	59.89%	-17.42%
pool6	0.027%	1024	59.76%	-17.55%

TABLE 3.2: Computation, storage, and accuracy trade-off with Latent Replay at different layers of a MobileNetV1 ConvNet trained continually on NICv2-391 with $RM_{size} = 1500$. Computation and instance size can be easily extrapolated from Table B.1 in the appendix, where the network architecture is exploded by reporting neurons, connections and weights at each layer.

For example, `conv5_4/dw` exhibits an interesting trade-off because the computation is about 32% of the native rehearsal one, the storage is reduced to 66% (more on this point in subsection 3.2.3) and the accuracy drop is mild (5.07%). CWR* (i.e. AR1* with latent replay layer $\equiv$ `pool6`) has a really negligible computational cost (0.027%) with respect to native rehearsal and still provides an accuracy improvement of $\sim$4% w.r.t. the non-replay case ($\sim$60% vs $\sim$56% as can be seen in Figure 3.7 and Figure 3.6, respectively).

Reducing Activations Storage in Latent Replay Even if in our CORe50 case study the external storage is quite limited (e.g., 1,500 $\times$ 32KB = 48 MB for latent replay at `conv5_4/dw`), scaling up to applications with thousands of classes could require storing much more activations and the external memory could become an issue. Fortunately, activations of layers in the upper part of the model can be sparsified, quantized, and encoded with almost no accuracy reduction. The authors of [47] show that MobileNetV1 activations can be compressed up to 10 times upon proper sparsification, encoding, and lossless entropy compression. In their experiments, even a moderate compression leads to a slightly improved accuracy because of the introduced regularization.

In the case of latent replay, we only need to sparsify the activations of the latent replay layer (and not of the entire network), potentially introducing a sort of information bottleneck. This can be easily achieved by adding an L1 term (with relative weight) to the loss function attracting toward zero the activations of the latent replay layer (see [47]). We performed some preliminary experiments to sparsify activations of layer `conv5_4/dw` during the first training experience starting from a

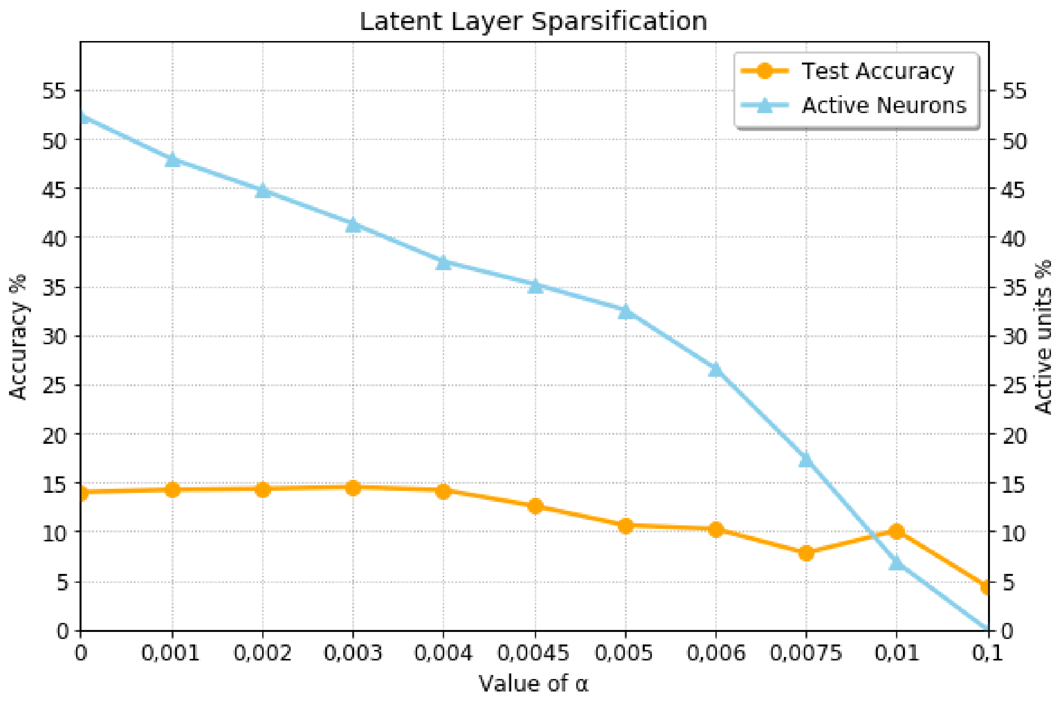

FIGURE 3.8: Sparsification of `conv5_4/dw` activations for different values of α and the corresponding accuracy after the first training batch.

non-sparsified ImageNet pre-trained model. Note that the weights of the latent replay layer and previous layers are frozen after the first training batch and no further sparsification can take place. The results are shown in Figure 3.8: for $\alpha = 0$ (i.e., no induced sparsification) ~52% of activations are non-zero due to the natural spasification effect of the Relu activation function and the accuracy is about 14%; as we increase α, the amount of non-zero activation start decreasing. Interestingly, for $\alpha = 0.004$ we can reduce the non-zero activations from ~52% to ~37% by achieving also a slight accuracy improvement (0.22%). By adding quantization and entropy encoding (out of the scope of this work) we believe that, analogously to [47], a $10\times$ compression is at reach with almost no accuracy loss.

Comparison with Other Approaches While the accuracy improvement of the proposed approach w.r.t. state-of-the-art replay-free techniques have been already discussed in previous sections, further comparison with other state-of-the-art continual learning techniques may be beneficial for better appreciating its practical impact and advantage. In particular, while AR1* and CWR* have been already proved to be substantially better than LWF and EWC on the NICv2-391 benchmark, a comparison with iCaRL, one of the best known replay-based techniques, is worth considering.

Unfortunately, iCaRL was conceived for incremental class learning scenarios and its porting to NIC (whose batches also include instances of know classes) is not straightforward. To avoid subjective modifications, we started from the code shared by the authors and emulated a NIC setting by: *(i)* always creating new virtual classes from instances in the coming experiences; *(ii)* fusing virtual classes together when evaluating accuracies. For example, let us suppose to encounter 300 instances of class 5 in batch 2 and other 300 instances of the same class in batch 7; while two virtual classes are created by iCaRL during training, when evaluating accuracy both classes

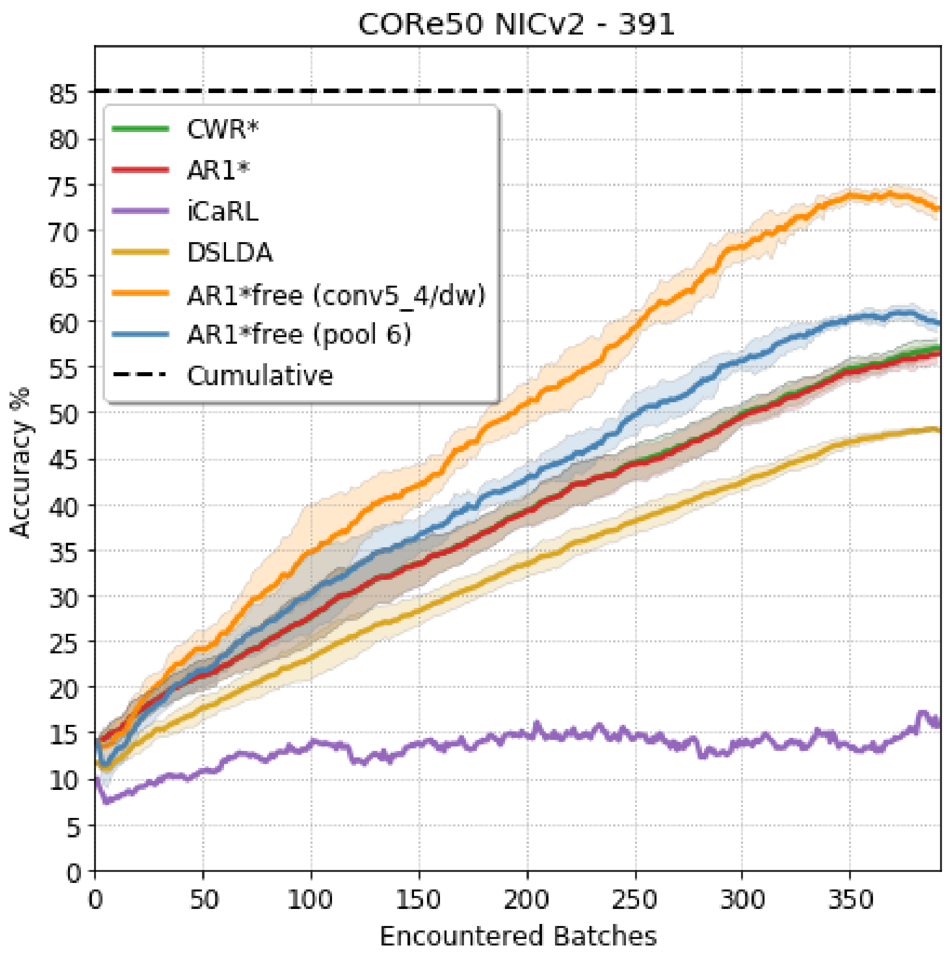

FIGURE 3.9: Accuracy results on the CORe50 NICv2-391 benchmark of CWR*, AR1*, DSLDA, iCaRL, AR1*free (`conv5_4`), and AR1*free (`pool6`). Results are averaged across 10 runs in which the batches order is randomly shuffled. Colored areas indicate the standard deviation of each curve. As an exception, iCaRL was trained only on a single run given its extensive run time (∼14 days).

point to the real class 5. The hereby modified iCaRL implementation, with an external memory of 8000 instances (much more than the 1500 used by the proposed latent replay, but in line with the settings proposed in the original paper [135]), was run on NICv2-391, but we were not able to obtain satisfactory results. In Figure 3.9 we report the iCaRL accuracy over time and compare it with AR1*free (`conv5_4/dw`), AR1* (`pool6`) as well as the top three performing replay-free strategies introduced in Section 3.1: CWR*, AR1* and DSLDA. While iCaRL exhibits better performance than LWF and EWC (as reported in Section 3.1.2), it is far from DSLDA, CWR*, and AR1*.

Furthermore, when the algorithm has to deal with a so large number of classes (including virtual ones) and training experiences, its efficiency becomes very low (as also reported in [103]). In Table 3.3 we also report the total run time (training and testing), memory overhead, and accuracy difference with respect to the cumulative upper bound. We believe AR1*free (`conv5_4/dw`) represents a good trade-off in terms of efficiency-efficacy with limited computational-memory overhead and only a ∼13% distance from the cumulative upper bound. For iCaRL, the total training

Strategy	Run Time (Minutes)	Mem. Overhead (Data+Params, MB)	Final Accuracy %	Δ Acc. % vs Cumulative
CWR*	21.4	0 + 0.2	56.99%	-28.27%
AR1*free (pool6)	23.7	5.8 + 12.4	59.75%	-25.51%
AR1*	39.9	0 + 12.4	56.32%	-28.94%
AR1*free (conv5_4/dw)	**41.2**	**48 + 0**	**72.23%**	**-13.03%**
DSLDA	79.1	0 + 0.2	48.02%	-37.24%
iCaRL	20185.0	375 + 0	15.65%	-69.61%

TABLE 3.3: Summary of the computation, memory, and accuracy trade-off for each strategy. Memory overhead includes both the data used for replay purposes as well as additional trainable parameters needed for continual learning. Each metric is averaged across 10 runs.

time was 14 days compared to a training time of less than 1 hour for the other techniques.

Experiments on OpenLORIS

In order to show the general applicability of latent replay in different continual learning settings, we also report and compare its performance on the OpenLORIS dataset, which has been used as the main benchmark in the recent IROS 2019 *"Lifelong Robotic Vision"* competition [128].

OpenLORIS is particularly interesting for continual learning in the context of robotic vision since its video sessions have been recorded on a real wheeled robot exploring its environment (see Figure 3.10). In this case, however, the scenario is quite different from CORe50 NICv2-391: it is based on a sequence of 12 relatively large experiences (∼14,000 samples each) containing only new examples of the same 69 classes made available in the first experience (that is, a benchmark belonging to the "New Instances" (NI) family, as described in Section 2.2). For this scenario:

- We use a MobileNetV2 [149] pre-trained on ImageNet-1k.
- We apply our *latent replay* approach to LWF [92], whose distillation steps proved to be effective to continually learn over a moderate number or large batches.

Seven finalists passed the first competition stage and submitted their solutions to the organizers who finally produced the scoreboard reported in Table 3.4. The accuracy of the proposed approach is just slightly lower than the top 1, but its inference time, replay memory, and model size are significantly better. Since the challenge evaluation criteria did not include specific metrics on training efficiency, unfortunately from this experiment we cannot appreciate the training efficiency of our solution.

3.2.4 Remarks on Latent Replay

In this section, it has been shown that latent replay is an efficient technique that can be used to continually learn new classes and new instances of known classes even from small and non-i.i.d. batches. State-of-the-art CL approaches such as AR1*, when extended with latent replay, can learn efficiently, and the achieved accuracy is not far from the cumulative upper bound (about 5% in some cases). The

OpenLORIS Challenge Results (7 finalists)				
Strategy	Final Acc. %	Inference Time (s)	Replay Size (Sample)	Model Size (MB)
SDU_BFA_PKU	99.56%	2,444.01	28,500	171.40
UniBo-Team (ours)	**97.68%**	**22.41**	**1,500**	**5.90**
HIK_ILG	96.86%	25.42	0	16.30
Vidit98	96.16%	112.2	13,000	9.40
NTU_LL	93.56%	4,213.76	0	467.10
Neverforget	92.93%	89.15	0	342.90
Guinness	72.9%	346.02	0	9.40

TABLE 3.4: Accuracy results at the end of the training and other metrics used for the OpenLORIS challenge benchmark.

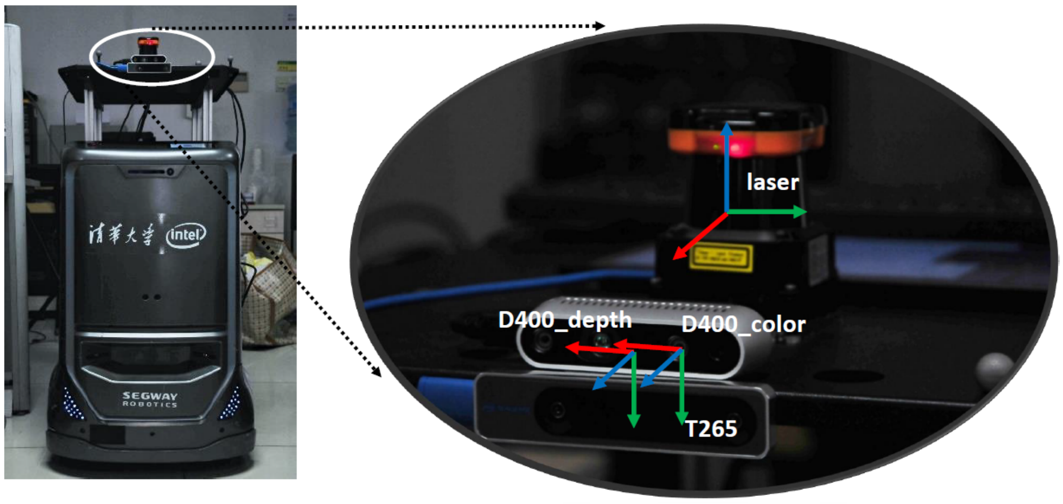

FIGURE 3.10: Wheeled robot used in the *Lifelong Robotic Vision* challenge at IROS 2019 [128] (left) and equipped with multiple sensors including two Real Sense cameras (right).

computation-storage-accuracy trade-off can be defined according to both the target application and the available resources so that even edge devices with no GPUs can learn continually from short videos. In fact, latent replay can enable continual learning at the edge on smartphone devices and ultra-low-power embedded boards as described in Sections 5.1 and 5.2 respectively.

Chapter 4

The Avalanche Library

A sustained interest in a certain research topic naturally leads to the creation of tools specifically designed to solve problems specific to that area. This reasoning can be generically applied to any research area.

Firstly, a huge effort is carried out by the early investigators. Apart from the heavy work concerning the initial theoretical efforts, they also need to implement their ideas. For research in the Computer Science field, that means implementing the idea as an experimental suite able to reproduce the expected results.

Secondly, a growing phase happens in which the research area is explored more extensively. During this phase, not only novel original solutions are designed, but the boundaries of the area are incrementally delimited. The rise of different theoretical solutions aimed at exploring the area in many original ways is a strong desiderata for any field of human knowledge. On the contrary, the proliferation of too many codebases, tools, and practices is hardly a desirable element.

Finally, the research area reaches a certain degree of maturity. This maturity leads to the creation of tools that implement and consolidate the best and most common practices.

For continual learning, the earliest efforts regarding the practical translation of research ideas into code concerned the implementation of elements working with (and often "around") mainstream Deep Learning frameworks such as Caffe, Theano-Lasagne, TensorFlow, and PyTorch. Following the publication of the first papers and the related open-sourced codebases, the community has some initial foundation on which subsequent works can be built. However, it is not obvious for research groups to adopt codebases written by other researchers. In fact, in a given field, if there is no consensus on the tools to use (if they even exist at that moment), each research group is usually tempted to implement their experimental suite from scratch, even if the same building blocks and mechanisms could be found in other open-sourced code bases. This has been quite evident in the continual learning field: during the growth stage of the continual learning area, many research groups opted for implementing their own solutions.

Apart from the lack of tools specifically designed for the task, the "roll-my-own" approach was made necessary for two main reasons:

- The adoption of a varied set of Deep Learning frameworks. At the time continual learning started its early growth phase, many frameworks were still competing for the role of mainstream tool for Deep Learning research (or they were still widely used). Caffe, Theano-Lasagne, TensorFlow, and PyTorch have been the most used ones in the continual learning landscape. As of now, this choice is much more restricted. Many libraries that were very popular until a few years ago are now less used if not completely discontinued. The choice of the Deep Learning framework to use is critical in the implementation of a continual learning code as it defines the degree of flexibility the researcher can count on. Starting from this premise, it is easy to identify the reason why each research group frequently implemented their ideas using different frameworks.

- The variety of the research landscape is another strong reason. The continual learning area touches many elements of computer science and neuroscience fields. With ambitious goals and little to no boundaries, research in the continual learning field quickly became very diversified in terms of goals, assumptions, benchmarks, and approaches. The diversification of research directions in a field is a good measure of its maturity. If the field is not mature enough, it is difficult to pinpoint the common building blocks and ideas upon which codebases are built. Each research direction may be based on different assumptions and thus require a different approach to implement the code to carry out experiments.

This made it difficult for continual learning researchers to adopt elements from other codebases and, because of that, no consensus on common elements could be reached. The choice of implementing a separate code base, each based on a different structure, idea, and philosophy, led to a fragmentation of the continual learning landscape. This, in turn, led to problems regarding the readability of the code and the reproducibility of experiments.

There is always a good reason for a researcher to delve into other researchers' code: to reproduce the results of experiments, to understand a specific technique so that it can be adapted and integrated into their novel solution, or even to just learn something new. In particular, for researchers, having their solution used and cited in other researchers' work is usually desirable. If the code accompanying a paper has strong technological barriers, then the work may even get underappreciated despite its theoretical soundness and good experimental results.

For instance, the first time the Latent Replay mechanism described in Section 3.2 was implemented, we decided to base the overall code on a customized version of Caffe. The choice of using Caffe allowed us to port that technique into the CORe Android application, which would not have been possible by using more modern libraries like TensorFlow or PyTorch (more on this in Section 5.1). However, this choice came at a cost: to make experiments reproducible, one had to manually install relevant dependencies and build the customized Caffe library from scratch. Considering the quite complicated setup involved in this pipeline, we decided to release a cuda-enabled dockerized version of the customized Caffe framework along with experiments code to make things easier for other researchers. Despite our efforts to make the code as accessible as possible by adding a comprehensive guide and helper scripts, the attempts of adopting the latent replay idea by other researchers were hindered by this technological barrier until a PyTorch version of it was released.

However, exceptions exist: the community might try to port particularly popular techniques to mainstream frameworks. However, these efforts may end up in implementations returning results not aligned with the original ones. To give another example, on a later occasion, we started an effort to port the iCaRL codebase for the Split-CIFAR experiments to PyTorch. We wanted to obtain an implementation perfectly aligned with the official one. However, while experiments on Split-ImageNet were implemented using TensorFlow, experiments on Split-CIFAR were implemented in Theano-Lasagne. Porting one deep learning code to a different library is not an easy task, especially if the starting codebase is based on legacy tools. Many defaults are usually overlooked: weight initialization, minibatch padding, bias initialization, how minibatches are ordered, how the batch normalization layers statistics are updated, etcetera. Each default may be different in each deep learning library. Differences in these defaults become very pronounced when porting a continual learning code, in which any minimal difference can lead to very different results.

Finally, in this fragmented landscape, things could have been even more complicated with the use of different programming languages. However, this problem did not arise for continual learning. Fortunately, the area started to get popular at a moment in which Python was well established as the reference language for machine learning and deep learning.

In the rest of this chapter, the desiderata and design principles for a continual learning library are discussed. This is followed by a description of the structure of the Avalanche library, a community effort born inside the ContinualAI organization.

Finally, many libraries were being built at the same time (between the years 2020 and 2021), usually with one team not knowing about the other's efforts. The most known alternatives are described in the last part (Section 4.3) of this chapter.

4.1 Libraries for continual learning research

The teams that led the efforts towards the design and implementation of the first general continual learning libraries started from the awareness that the continual learning research field reached a sufficient degree of maturity and, from a code perspective, that the degree of fragmentation had become unmanageable. These are the reasons why, during the second half of my Ph.D., we (the core Avalanche team) put a huge effort into the design and implementation of the Avalanche library.

Writing a continual learning library is not an easy task. The continual learning research landscape is very variegated: each direction and research niche is based on its own set of abstractions. However, many elements are common to all continual learning sub-areas (and thus codebases), which means that a comprehensive library able to organize these building blocks organically is highly desirable. However, libraries may come in different flavors, they may be designed with different ideas in mind, different philosophies, etcetera.

For instance, should the library be created by expanding an existing codebase released for a paper, or should it be created from scratch? Should the library be focused to solve a specific problem, or should it try to cover all aspects of continual learning research? In addition, should the library pose constraints on how it is used? Many libraries force a framework-alike vision, in which the user is forced to use the library in a strict predefined way, while other libraries are more similar to a collection of tools the user can freely use. And, of course, many "in-betweens" exist. Should the library be tailored exclusively for research purposes or to a "research plus production" one? Finally, the group of users who will use the library should be clearly defined from the beginning: will the library be used internally (in a research group, institution, etcetera), or is it aimed at the general public?

Writing a library for such a wide research area requires a lot of effort. Starting with clear ideas about the aforementioned elements is essential for the success of the effort. For Avalanche, we decided to implement the library from scratch with the idea of creating a library for the general public. In fact, Avalanche was designed to provide a shared and collaborative code base for fast prototyping, training, and reproducible evaluation of continual learning algorithms. However, to answer the other questions, one must first have in mind a well-defined list of desiderata for the library. Based on those, one can easily extract the design principles that guide the design and implementation processes.

4.1.1 Desiderata and Design Principles

Avalanche has been designed with five main principles in mind: i) *Comprehensiveness and Consistency*; ii) *Ease-of-Use*; iii) *Modularity (and Independence)*; iv) *Modularity (and Independence)*; iv) *Reproducibility and Portability*; v) *Efficiency and Scalability*. These are important for any continual learning project, but they become essential for tackling the most interesting research challenges and real-world applications.

Comprehensiveness The Avalanche library was created to be as general as possible. In other words, a desideratum for a continual learning library is to support as many research directions as possible, including the most exotic approaches. This is the reason why the main design principle for Avalanche follows from the concept of *comprehensiveness*, the idea of providing an exhaustive and unifying library with *end-to-end* support for continual learning research and development. A comprehensive codebase does not only provide a unique and clear access point to researchers and practitioners working on the topic, but also favors *consistency* across the library, with a coherent and easy interaction across modules and sub-modules. Last but not least, it promotes the consolidation of a large community able to provide better support for the library.

Modularity In practice, apart from the goal of supporting many research directions, the comprehensiveness principle has an additional implication: to be comprehensive, the library should cover all the macro-blocks usually involved in a research code: data loading, model definition, training, evaluation, and logging. This is why *modularity* is another fundamental design principle (and desideratum). In Avalanche, simplicity is sometimes bent in favor of modularity and reusability. This is essential for scalability and to collaboratively bring the codebase to maturity.

Ease-of-Use The third desideratum is *simplicity*: to offer simple solutions to complex problems and to enable for a simple usage of the library. This desideratum is difficult to translate into a proper design principle.

The ease of use may be heavily undermined when posing strict constraints on how the library must be used. This is something that can easily happen when designing a library that has to cover all the parts of the codebase. For instance, design decisions regarding the training module may introduce heavy constraints on how the user should use the evaluation or logging modules.

Combining these first three desiderata (*comprehensiveness*, *modularity*, and *ease-of-use*) is a particularly difficult task. This is why a particular focus on module *independence* is maintained through Avalanche to guarantee the stand-alone usability of individual module functionalities. This design principle also facilitates the learning of a particular module by new users. In addition, we concentrated our efforts on the design of an intuitive Application Programming Interface (API), an official website, and rich documentation with a curated list of executable notebooks and examples[1].

Reproducibility and Portability Reproducing research paper results is a difficult but much needed task in machine learning [69]. The problem is exacerbated in continual learning. A critical design objective of Avalanche is to allow experimental results to be seamlessly reproduced. This allows researchers to simply integrate their own original research into the shared codebase and compare their solution with the existing literature, forming a virtuous circle. Hence, reproducibility is not only a core objective of sound and consistent scientific research but also a means to speed up the development of original continual learning solutions.

Efficiency and Scalability Computational and memory requirements in machine learning have grown significantly throughout the last two decades [170]. Standard deep learning libraries such as TensorFlow [105] or PyTorch [120] already focus on *efficiency* and *scalability* as two fundamental designing principles, since modern research experiments can take months to complete [150]. Avalanche is based on the same principles: offering the end-user a seamless and transparent experience regardless of the use case or the hardware platform that the library is run on.

Other implied desiderata Given the aforementioned five fundamental points, it is worth describing a few other desiderata directly derived from those.

Neutral terminology and naming conventions A desirable element in a continual learning library is the use of neutral terminology and naming conventions. The decisions we made for Avalanche are described in more detail in Section 4.2.1. This has practical implications in the naming of modules, classes, methods, etcetera.

Easy to extend, readability of internals Avalanche is strongly modular not only to allow for easier use of the library but also to allow users to extend the library for their needs. However, extendability should be conceived at the design level (modularity alone is not enough). In other words, extending the library should not require the user to create a separate, customized copy of the library. In addition, extendability shouldn't happen by implementing workarounds. Instead, a properly designed library should already expose mechanisms to allow for its extension. This is why Avalanche was designed to cover all mainstream research directions and, for areas in which it fails to do so, to provide mechanisms to adapt and extend the library behavior. This is highly desirable, especially in the continual learning field, where the research landscape is constantly evolving.

[1] The official website, documentation, notebooks, and examples are available at `https://avalanche.continualai.org`.

Implement best practices, but allow for customization Finally, the user should be able to fully customize the training flow. While a library may expose a recommended template that brings together the best practices of the field, the user should be able to customize every aspect of it.

Community-driven collaborative effort Another, non technical, fundamental desiderata should be considered. Developing and maintaining an open-source project usually implies some way to collaborate with users of the project so that external contributions can be accepted to fix bugs and implement new features. However, this is not mandatory. Many open-source projects may just be carried out by a closed development team. For a continual learning library, the latest approach would mark its demise. Continual learning is a very variegated and continually evolving research area, which means that any continual learning library that doesn't receive a constant flow of contributions from the community will almost surely be lagging behind in terms of supported benchmarks, models, training paradigms, etcetera.

This is why the Avalanche project was conceived as a community-driven, collaborative project. The project is powered and maintained by ContinualAI, a non-profit research organization and, arguably, the largest open community on Continual Learning for AI.

To allow for an effective contribution flow from the community, the project is mainly developed by a core team, with each member of the team in charge of maintaining a single module. Contributions, questions, and issues about a specific module are handled by the reference maintainer. At the same time, library-wide changes are discussed by all team members to allow for changes to be consistent across the project.

4.2 Avalanche: a comprehensive CL library

Avalanche is an open-source (MIT licensed) end-to-end library for continual learning based on PyTorch [120], devised to provide a shared and collaborative codebase for fast prototyping, training, and evaluation of continual learning algorithms.

4.2.1 Continual Learning Framework

Recently, we have witnessed a significant attempt to formalize a general (theoretical) framework for continual learning algorithms [96, 98, 177]. These proposals often categorize scenarios and algorithms based on their unique properties and specific settings. However, as as just mentioned, within the formal design of Avalanche, we take a different approach.

Given the fast-evolving, often conflicting views of the problem, we aimed to lower the number of assumptions to a minimum, favoring simplicity and flexibility. In practice, this translates into providing users with a set of building blocks that can be used in any continual learning solution without imposing any strong nomenclature, constraining abstractions, or assumptions.

In Avalanche, data is modeled as an ordered sequence (or stream) composed of n, usually *non-iid*, learning *experiences*:

$$e_0, e_1, \ldots, e_n.$$

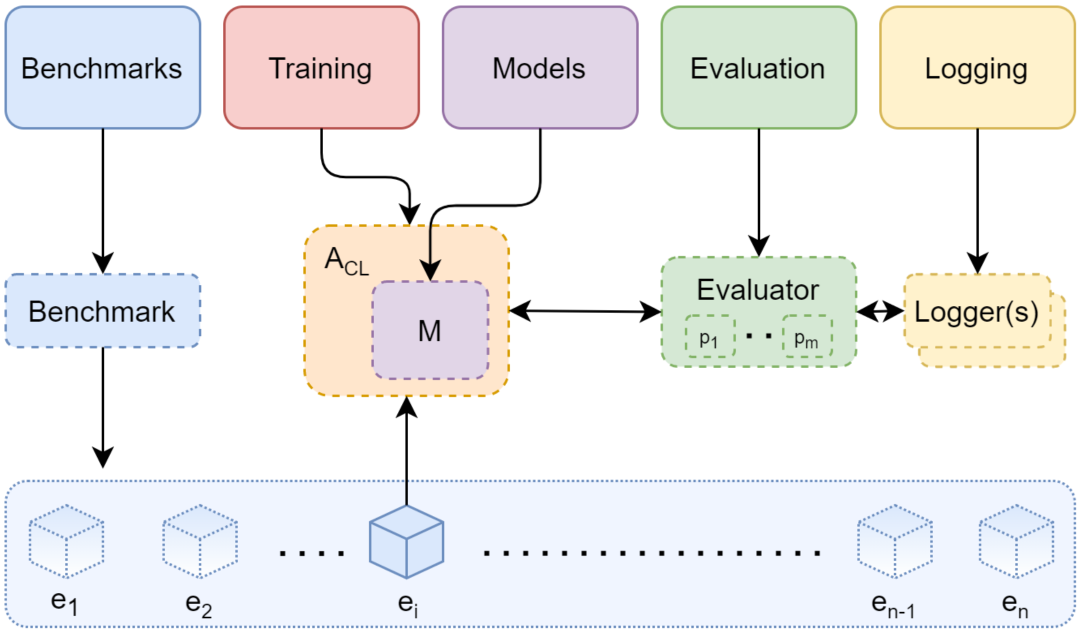

FIGURE 4.1: Operational representation of Avalanche with its main modules (top), the main object instances (middle) and the generated stream of data (bottom). A *Benchmark* generates a stream of experiences e_i which are sequentially accessible by the continual learning algorithm A_{CL} with its internal model M. The *Evaluator* object directly interacting with the algorithm provides a unified interface to control and compute several performance metrics (p_i), delegating results logging to the *Logger(s)* objects.

A learning experience is a set composed of one or multiple samples which can be used to update the model. This is often referred to in the literature as *batch* or *task*. This formulation is general enough to be used in several continual learning contexts, such as supervised, reinforcement, or unsupervised continual learning. Avalanche provides a general set of abstractions that do not impose any particular constraints on the content of the experiences. For example, in a supervised training regime, each learning experience e_i can be seen as a set of triplets $\langle x_i, y_i, t_i \rangle$, where x_i and y_i represent an input and its corresponding target, respectively, while t_i is the *task label*, which may or may not be available.

During training, a *continual learning algorithm* A_{CL} processes experiences sequentially and uses them to update the model and its internal state. In Avalanche, each algorithm has a training mode, used to update the model, and an evaluation mode, which may be used to process streams of experiences for testing purposes.

The continual learning framework we propose can be formalized as follows.

Definition (Continual Learning Framework)**.** A continual Learning algorithm A_{CL} is expected to update its internal state (e.g. its internal model M or other data structures) based on the sequential exposure to a non-stationary stream of experiences $(e_1, \ldots, e_n)$. The objective of A_{CL} is to improve its performance on a set of metrics $(p_1, \ldots, p_m)$ as assessed on a test stream of experiences $(e_1^t, \ldots, e_n^t)$.

4.2.2 Main Modules

The library is organized into five main modules: **Benchmarks** (Section 4.2.2), **Training** (Section 4.2.2), **Evaluation** (Section 4.2.2), **Models** (Section 4.2.2), and **Logging**

	Supported features
Benchmarks	Split/Permuted/Rotated MNIST [99], Split Fashion Mnist [40], Split Cifar10/100/110 [135, 103], Split CUB200, Split ImageNet [135], Split TinyImageNet [31], Split/Permuted/Rotated Omniglot [151], CORe50 [95], OpenLORIS [158], Stream51 [141], Split-iNaturalist [31], CTrL [178], Endless-Continual-Learning Simulator [63].
Scenarios	Multi Task [87], Single Incremental Task [87], Multi Incremental Task [87], Class incremental [135, 177], Domain Incremental [177], Task Incremental [177], Task-agnostic, Online, New Instances, New Classes, New Instances and Classes.
Strategies	Naive (Finetuning), Replay, Joint Training, CWR* [96], GDumb [126], Cumulative, LwF [92], GEM [99], A-GEM [20], EWC [79], Synaptic Intelligence [189], AR1 [103], Less-Forgetful Learning [75], Gradient-Based Sample Selection (replay) [4], CoPE [29], iCarl [135], DeepSLDA [59].
Metrics	Accuracy, Loss (user specified), Confusion Matrix, Forgetting, CPU Usage, GPU usage, RAM usage, Disk Usage, Timing, Multiply, and Accumulate [72, 35].
Loggers	Text Logger, Interactive Logger, CSVLogger, Tensorboard Logger [105], Weights & Biases Logger [11].

TABLE 4.1: Avalanche supported features for the Beta release (v0.1.0).

(Section 4.2.2). Table 4.1 summarizes the features provided by Avalanche at the current stage of development. In Figure 4.1, an operational representation of the library modules and their interplay within the aforementioned reference framework is shown.

Benchmarks

Continual learning revolves around the idea of dealing with a non-stationary stream of experiences. An example stream from the standard SplitMNIST benchmark [189] composed of five experiences is shown in Figure 4.2. A target system powered by a continual learning strategy is required to learn from experiences (e.g., by considering additional classes in a class-incremental setting [106]) in order to improve its performance or expand its set of capabilities. This means that the component in charge of generating the data stream is usually the first building block of a continual learning experiment. It is no surprise that a considerable amount of time is spent defining and implementing the data loading module. The benchmarks module offers a powerful set of tools one can leverage to greatly simplify this process.

The term *benchmark* is used in Avalanche to describe a recipe that specifies how the stream of data is created by defining the originating dataset(s), the contents of the stream, the amount of examples, task labels, boundaries [3], etcetera. When defining such elements, some degree of freedom is retained to allow obtaining different *benchmark instances*. For example, different instantiations of the SplitMNIST benchmark [189] can be obtained by setting different class assignments. Alternatively, distinct instances of the PermutedMNIST [50] benchmark can be obtained by choosing different pixel permutations.

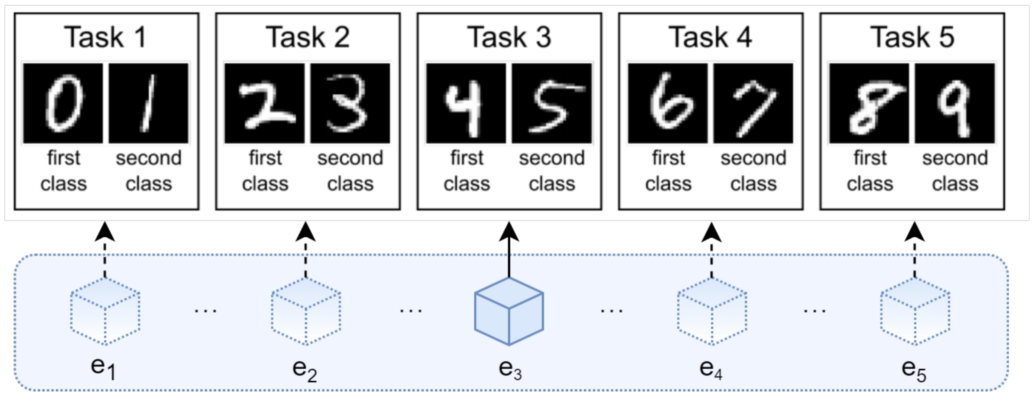

FIGURE 4.2: Example of a generated stream in Avalanche, composed by five experiences, implementing the common SplitMNIST benchmark [189]. When accessing experience e_3, the continual learning algorithm has no access to previous or future experiences.

The *benchmarks* module is designed with the idea of providing an extensive set of out-of-the-box loaders covering the most common benchmarks (i.e. SplitCIFAR [135], PermutedMNIST [50], etc.) through the *classic* submodule. A simple example illustrating how to use the "SplitMNIST" benchmark [189] is shown in Figure 4.3. Moreover, a wide range of tools to allow for the creation of customized benchmarks is available. The goal is to provide full support to researchers implementing benchmarks that do not easily fit into the existing categories.

Most out-of-the-box benchmarks are based on the dataset implementation provided by the `torchvision` library. A proper implementation is provided for other datasets (such as CORe50 [95], Stream-51 [141], and OpenLORIS [158]). The benchmark preparation and data loading process can seamlessly handle memory-intensive benchmarks, such as Split-ImageNet [135], without the need to load the whole dataset into memory in advance.

Further, the benchmarks module is entirely standalone, meaning that it can be used independently from the rest of Avalanche.

Benchmarks creation The benchmarks module exposes a uniform API that makes it easy to define a new continual learning benchmark.

Classic Benchmarks

```python
benchmark_instance = SplitMNIST(
    n_experiences=5, seed=1)
    # Other useful parameters
    # return_task_id=False/True
    # fixed_class_order=[5, 0, 9, ...]
```

FIGURE 4.3: Simple instantiation of a *Classic* continual learning benchmark.

The *classic* package hosts an ever-growing set of common benchmarks and is expected to cover the usage requirements of the vast majority of researchers. However, there are situations in which implementing a novel benchmark is required. Avalanche offers a flexible API that can be used to easily handle this situation.

Starting from the higher-level API, Avalanche offers explicit support for creating benchmarks that fit one of the ready-to-use scenarios. The concept of *scenario* is slightly different from that of 'benchmark' as it describes a more general recipe independent of a specific dataset. If the benchmark to be implemented fits either in the *New Instances* or *New Classes* scenarios [103], one can consider using one of the specific generators `nc_scenario` or `ni_scenario`. Both generators take a pair of train and test datasets and produce a benchmark instance. The New Classes generator splits all the available classes in a number of subsets equal to the required number of experiences. Patterns are then allocated to each experience by assigning all patterns of the selected classes. This means that the New Classes generator can be used as a basis to set up Task or Class-incremental learning benchmarks [177]. The New Instances generator splits the original training set by creating experiences containing an equal amount of patterns using a random allocation schema. The main intended usage for this generator is to help in setting up Domain-Incremental learning benchmarks [177]. Most classic benchmarks are based on these generators. Figure 4.4 shows a simple example of using `nc_scenario`.

Benchmarks Generators

```python
# Nearly all datasets from torchvision
# are supported

mnist_train = MNIST('./mnist', train=True)
mnist_test = MNIST('./mnist', train=False)
benchmark_instance = nc_scenario(
    train_dataset=mnist_train,
    test_dataset=mnist_test,
    n_experiences=n_experiences,
    task_labels=True/False)
```

FIGURE 4.4: Example of using the *"New Classes"* benchmark generator on the MNIST dataset.

If the benchmark does not fit into a predefined scenario, a *generic generator* can be used. At the moment, Avalanche allows users to create benchmark instances from lists of files, Caffe-style filelists [73], lists of PyTorch datasets, or even directly from Tensors. We expect that the number of generic generators will rapidly grow in order to cover the most common use cases and allow for maximum flexibility.

Streams and Experiences Not all continual learning benchmarks limit themselves to describing a single stream of data. Many of them contemplate an out-of-distribution stream, a validation stream, and possibly several other arbitrary streams, each linked to a different semantic. For instance, [20] proposes a benchmark where a separate stream of data is used for cross-validation, while [141] defines an out-of-distribution stream used to evaluate the novelty detection capabilities of the model.

Motivated by this remark, we decided to model benchmark instances as a composition of streams of experiences. This choice has two positive effects on the resulting API. Firstly, the way streams and experiences can be accessed is shared across all benchmark instances. Secondly, this modeling of benchmark instances does not force any preconceived schema upon researchers. Avalanche leaves the semantic aspects regarding the definition and usage of each stream to the creator of the benchmark.

A simple example showing the versatility of this design choice concerns the test stream: in order to allow for a proper evaluation of a continual learning strategy, benchmarks do not only need to describe the stream of training experiences but also need to give a proper description of a evaluation protocol. Such protocol is, in turn, based on one or more test datasets on which appropriate metrics can be computed. In many cases, the test data may need to be structured into a sequence of 'test experiences', analogously to what happens with the training data stream.

For instance, in Class-Incremental learning, the test set may be split into different experiences, each containing only test instances related to classes encountered in the corresponding training experience.

Avalanche currently supports two main streams: *train* and *test*, and arbitrary streams (for instance, out-of-distribution stream, validation stream, etcetera) can be easily added when needed.

Each experience can be obtained by iterating over one of the available streams. Figure 4.5 shows how, starting from a benchmark instance, streams can be retrieved and used. Each experience carries a PyTorch dataset, task label(s), and other useful benchmark-specific information that can be used for introspection. An experience also carries a numerical identifier that defines its position in the originating stream. In fact, experiences in a stream can be also accessed by index. This functionality makes it easy to couple related experiences from different streams.

Main Training Loop

```python
train_stream = benchmark_instance.train_stream
test_stream = benchmark_instance.test_stream

for idx, experience in enumerate(train_stream):
    dataset = experience.dataset

    print('Train dataset contains',
        len(dataset), 'patterns')

    for x, y, t in dataset:
        ...

    test_experience = test_stream[idx]
    cumulative_test = test_stream[:idx+1]
```

FIGURE 4.5: Example of the main training loop over the stream of experiences.

Task Labels and Nomenclature Every mechanism, internal aspect, name of function and class in the benchmarks module were designed with the intent of keeping Avalanche as neutral as possible with respect to the presence of task labels. Task boundaries, task descriptors and task labels are widely used in the continual learning literature to define both semantic and practical aspects of a benchmark. However, the meaning of those concepts is usually blurred with the definition of the specific benchmark to which they are applied to, making it hard to clearly pin-point a generic way to manage them.

Based on this observation, and due to the fact that the usage of task-specific information may become more extravagant or sophisticated in the future, we decided that

Avalanche should not force any kind of convention. This means that the choice of whether to use task labels and how to use them is left to the user.

Following this idea, the `GenericCLScenario` class, which is the common class for all scenarios instances, allows researchers to assign task labels at pattern granularity, thus allowing for experiences with zero or more task labels. We deemed this the most natural choice for Avalanche: we believe that a continual learning library should not constrain researchers by superimposing a certain view of the field upon them. Instead, the idea of enabling the user to create complex setups in a simple way, without forcing a subjective interpretation, will probably prove to be more robust as the field continues to evolve.

Training

The `training` module implements both popular continual learning strategies and a set of abstractions that make it easy for a user to implement custom continual learning algorithms. Each strategy in Avalanche implements a method for training (`train`) and a method for evaluation (`eval`), which can work either on single experiences or on entire slices of the data stream. Currently, Avalanche provides 17 continual learning strategies (with many more to come), that can be used to train baselines in a few lines of code, as shown in Figure 4.6. See Table 4.1 for a complete list of the available strategies.

Training Strategies

```
strategy = Replay(model, optimizer,
                  criterion, mem_size)
for train_exp in scenario.train_stream:
    strategy.train(train_exp)
    strategy.eval(scenario.test_stream)
```

FIGURE 4.6: Simple instantiation of an already available strategy in Avalanche.

Training/Eval Loops In Avalanche, continual learning strategies subclass `BaseStrategy`, which provides generic training and evaluation loops. These can be extended and adapted by each strategy. For example, `JointTraining` implements offline training by concatenating the entire data stream in a single dataset and training only once. The pseudo-code in Figure 4.7 shows part of the `BaseStrategy.train` loop (`eval` has a similar structure).

Under the hood, `BaseStrategy` deals with most of the boilerplate code. The generic loops are able to seamlessly handle common continual learning scenarios, independently of differences such as the presence or absence of task labels.

Plugin System `BaseStrategy` provides a simple callback mechanism. This is used by strategies, metrics, and loggers to interact with the training loop and execute their code at the correct points using a simple interface and provides an easy interface to implement new strategies by adding custom code to the generic loops. `BaseStrategy` provides the global state (current mini-batch, logits, loss, . . .) to suitable plugins so that they can access all the information they need. In practice, most strategies in Avalanche are implemented via plugins. This approach has several advantages compared to a custom training loop. Firstly, the readability of the code is

Training Structure

```
def train(experiences):
        before_training()
        for exp in experiences:
                train_exp(exp)
        after_training()

def train_exp(exp):
        adapt_train_dataset()
        make_train_dataloader()
        before_training_exp()
        for epoch in range(n_epochs):
                before_training_epoch()
                training_epoch()
                after_training_epoch()
        after_training_exp()
```

FIGURE 4.7: Main training structure, the skeleton of the `BaseStrategy` class.

enhanced since most strategies only need to specify a few methods. Secondly, this allows for multiple strategies to be combined together. For example, we can define a hybrid strategy that uses Elastic Weight Consolidation (EWC) [79] and replay using the snippet of code shown in Figure 4.8.

Hybrid Strategies

```
replay = ReplayPlugin(mem_size)
ewc = EWCPlugin(ewc_lambda)
strategy = BaseStrategy(
    model, optimizer,
    criterion, mem_size,
    plugins=[replay, ewc])
```

FIGURE 4.8: Example of an on-the-fly instantiation of hybrid strategies through Plugins.

Evaluation

The performance of a CL algorithm should be assessed by monitoring multiple aspects of the computation [87]. The `evaluation` module offers a wide set of metrics to evaluate experiments.
Avalanche's design principle is to separate the issues of *what to monitor* and *how to monitor* it: the `evaluation` module provides support for the former through the metrics, while the `logging` module fulfills the latter (Section 4.2.2). Metrics do not specify in which format their output should be displayed, while loggers do not alter metrics logic. Metrics can work even without a logger: the strategy's *train* and *eval* methods return a dictionary with all the metrics logged during the experiment.

Few lines of code are sufficient to monitor a vast set of metrics: *accuracy*, *loss*, *catastrophic forgetting*, *confusion matrix*, *timing*, *ram/disk/CPU/GPU usage* and Multiply and Accumulate [72] (which measures the computational cost of the model's forward pass in terms of floating point operations). Each metric comes with a standalone class and a set of plugin classes aimed at emitting metric values on specific moments during training and evaluation.

Stand-alone Metrics Stand-alone metrics are meant to be used independently of all Avalanche functionalities. Each metric can be instantiated as a simple Python object. The metric will keep an internal state to store metric values. The state can be reset, updated or returned to the user by calling the related `reset`, `update` and `result` methods, respectively.

Plugin Metrics Plugin metrics are meant to be easily integrated into the Avalanche training and evaluation loops. Plugin metrics emit a curve composed by multiple values at different points in time. Usually, plugin metrics emit values after each training iteration, training epoch, evaluation experience or at the end of the entire evaluation stream. For example, `EpochAccuracy` reports the accuracy over all training epochs, while `ExperienceLoss` produces as many curves as the number of experiences. Each curve monitors the evaluation accuracy of an experience at the end of each training loop. Avalanche recommends the use of already implemented helper functions to simplify the creation of each plugin metric. The output of these functions can be passed as parameters directly to the `EvaluationPlugin`.

Evaluation Plugin `EvaluationPlugin` is the component responsible for the orchestration of both plugin metrics and loggers. Its role is to gather metrics outputs and forward them to the loggers during training and evaluation loops. All the user has to do to keep track of an experiment is to provide the strategy object with an instance of the `EvaluationPlugin` with the target metrics and loggers as parameters. Figure 4.9 shows how to use the evaluation plugin and metric helper functions.

Avalanche's effort to monitor different facets of performance aims at enabling a wider experimental assessment, which is too often focused only on the forgetting of previous knowledge [35].

Evaluation Plugin

```
eval_plugin = EvaluationPlugin(
    accuracy_metrics(experience=True),
    loss_metrics(minibatch=True, stream=True),
    forgetting_metrics(experience=True),
    timing_metrics(minibatch=True),
    gpu_usage_metrics(gpu_id, epoch=True),
    loggers=[InteractiveLogger(),
             TextLogger(open('out.txt', 'w')),
             TensorboardLogger()])
```

FIGURE 4.9: Avalanche evaluation plugin (or *evaluator*) object instantiation example.

Logging

Nowadays, logging facilities are fundamental to monitor in real time the behavior of an ongoing experiment (which may last from minutes to days). The Avalanche `logging` module is in charge of displaying to the user the result of each plugin metric during the different experiment phases. Avalanche provides five different loggers: `TextLogger`, `InteractiveLogger`, `CSVLogger`, `TensorboardLogger` [105], and `WandBLogger` (Weights & Biases) [11]. These loggers can be used to write detailed reports to textual files, standard output, and dashboards. As soon as a metric emits a value, the Text Logger prints the complete metric name followed by its value in

the destination file. The `InteractiveLogger` reports the same output as `TextLogger`, but it also uses the *tqdm* package[2] to display a progress bar during training and evaluation. `TensorboardLogger` and `WandBLogger` are able to show images and more complex outputs, which cannot be appropriately printed on standard output or textual files.
Integrating loggers into both training and evaluation loops is straightforward. Once created, loggers have to be passed to the `EvaluationPlugin`, which will be in charge of redirecting metrics outputs to each logger during the experiment. See Figure 4.9 for an example of loggers creation.

Users can easily design their own loggers by extending the class `StrategyLogger`, which provides the necessary interface to interact with the `EvaluationPlugin`.

Models

The Avalanche `models` module offers a set of simple machine learning architectures ready to be used in experiments. The module contains versions of feedforward and convolutional neural networks. The main purpose of these models is to let the user focus on Avalanche features, rather than on writing lines of code to build a specific architecture.

4.2.3 Next steps

Avalanche aims to provide a coherent, end-to-end, easily extendable library for continual learning research and development; a solid reference point and shared resource for researchers and practitioners working on continual learning and related areas.

As reported in Table 4.1, the current Avalanche `Beta` version focuses on continual supervised learning for vision tasks, as a significant amount of deep learning research in this area was designed and assessed in this context [55]. However, being Avalanche a *community-driven* effort, the plan in both the near and medium terms is to support the integration of additional learning paradigms (e.g. Reinforcement or Unsupervised Learning), tasks types (e.g. Detection, Segmentation), application contexts (e.g. Natural Language Processing, Speech Recognition), and problem type (Object Detection, Segmentation,) depending on the research community demands and priorities.

4.3 Alternatives

Reproducibility is one of the main principles upon which Avalanche is based. Experiments in the continual learning field are often challenging to reproduce due to the different implementations of protocols, benchmarks, and strategies by different authors. This issue of insufficient reproducibility is not limited to continual learning. The whole artificial intelligence community is affected; a number of authors have recently discussed some possible solutions to the problem [56, 124, 131].

The advent of machine (and deep) learning libraries, mainly TensorFlow [105] and PyTorch [120] has partially mitigated the reproducibility problem.

[2]`https://tqdm.github.io`

Using these libraries assures a standard implementation of many machine learning building blocks, reducing ambiguities due to bespoke and different implementations of basic concepts. However, this proved to be insufficient to cover the needs of the research community working in the continual learning area.

The year 2020 has been a turning point for the birth of libraries specifically aimed at the continual learning area. Before that, the community has put a lot of effort into addressing the aforementioned problems, by providing code and libraries aimed to increase the reproducibility of continual learning experiments [31, 68, 106, 152, 176, 177]. On the one hand, these first attempts lack the generality and the consistency of Avalanche and, in general, of a library designed with generality in mind, especially regarding the creation of different and complex benchmarks, and the continual support of a large community. On the other hand, they demonstrated the growing interest of the entire community in these issues.

Another area other than continual learning that has recently seen a proliferation of libraries and tools similar in spirit to Avalanche is reinforcement learning (RL). One of the most popular such benchmark RL libraries is OpenAI Gym [14], within which a multitude of different RL environments is available. A similar library is ViZDoom [184], in which an agent plays the famous computer game *Doom*. Other relevant projects in the field of reinforcement learning are Dopamine [19], which focuses on simplicity and easy prototyping, and project Malmo [74], which is based on the famous Minecraft game. Many of these libraries, however, only focus on the agent's interaction with the environment. This problem is addressed by other libraries that include standard implementations of baselines algorithms, such as OpenAI baselines [34] and stable baselines [64].

Another prominent example of a collection of baseline training strategies and pretrained models is the natural language processing transformers library by Hugging Face [183]. Many basic concepts upon which Avalanche is based (e.g. plugins, loggers, benchmarks) can also be found in more general machine learning libraries such as PyTorch Lighting [39] and fastai [67].

Another important problem in research is the bookkeeping of experiments. As discussed in Section 4.2.2, Avalanche already implements a fine-grained and punctual logging, which allows to visualize and save the results of different experiments. Moreover, Avalanche could be easily integrated with standalone libraries specifically developed for experiments bookkeeping and visualization, such as Sacred [52] or Weights and Biases (wandb) [11].

The motivations and the community needs behind the development of Avalanche were reinforced by the recent publication of similar continual learning libraries. Continuum[3] [36] focuses on data loading and processing, and could be related to the functionalities provided by the *benchmark* module of Avalanche. Sequoia[4] [114] is based on Continuum for the data loading but, in addition, it provides a playground for research at the intersection of Continual, Reinforcement, and Self-Supervised Learning.

As of now, Continuum and Sequoia are the main alternatives to Avalanche. Differently from the other aforementioned efforts, these libraries share with Avalanche

[3]`https://github.com/Continvvm/continuum`
[4]`https://github.com/lebrice/Sequoia`

the idea of creating a comprehensive library that could cover the needs of a very variegated set of research directions.

4.3.1 Continuum

Continuum [36] mainly focuses on the data loading aspect of the continual learning pipeline. The goal is to provide many pre-implemented data loaders (benchmarks) to be used out-of-the-box, as well as plain PyTorch dataset implementations. Continuum also provides a set of pre-implemented metrics such as accuracy-based ones (accuracy over time, average incremental accuracy) and more continual learning-specific ones such as *forward transfer* and *backward transfer* [99]. It also includes a basic logging facility able to export relevant metrics, as well as helper mechanisms to manage the replay buffer (including the herding method described in iCaRL [135]).

Continuum has been designed with a focus on simplicity, efficiency, and extendability. Custom benchmarks, metrics, loggers and replay management mechanisms can be easily implemented and customized by overriding a set of base classes. In addition, Continuum supports simple generators for the New Classes (NC), New Instances (NI), and New Instances and Classes (NIC) scenarios (described in Chapter 2).

Continuum shares a lot of similarities with Avalanche, especially on the benchmark creation aspects and on how extendability can be achieved by the users. However, Avalanche features a more comprehensive framework that extensively covers the training aspects of continual learning experiments. Due to its simplicity, Continuum is an excellent base on which a more complex codebase can be built. In fact, Sequoia uses Continuum under the hood to manage (or even completely outsource) certain aspects of the benchmark creation process.

Getting started example of Continuum

```
dataset = MNIST("my/data/path", download=True, train=True)
scenario = ClassIncremental(
    dataset,
    increment=1,
    initial_increment=5
)

for task_id, train_taskset in enumerate(scenario):
    train_taskset, val_taskset = split_train_val(train_taskset, val_split=0.1)
    train_loader = DataLoader(train_taskset, batch_size=32, shuffle=True)
    val_loader = DataLoader(val_taskset, batch_size=32, shuffle=True)

    for x, y, t in train_loader:
        # Run training
```

FIGURE 4.10: *Getting started* example of Continuum, showing the creation and use of the Split-MNIST benchmark.

4.3.2 Sequoia

Sequoia [114] is an ambitious project born with the idea of providing a comprehensive framework able to cover both more common (supervised learning and classification) and less explored (continual reinforcement learning, semi-supervised learning, etcetera) areas of the continual learning research. Sequoia has been designed following a sound theoretical framework [114] that proposes a hierarchical organization of research settings. This hierarchy is concretely implemented as a class tree.

Differently from Continuum (and similarly to Avalanche), Sequoia covers all aspects of continual learning experiments: benchmark creation, training, metrics, and logging. On the benchmark side, Sequoia relies on Continuum to cover the dataset implementation. On the training side, Sequoia features various out-of-the-box *native* strategies. At the same time, Sequoia was designed to be very flexible: because of this, it can use strategies from Avalanche and Stable Baselines3 [132].

Sequoia offers a comprehensive theoretical and practical framework for continual learning research. As of now, Avalanche is not able to cover all the settings managed by Sequoia, although efforts towards adding support for Continual Reinforcement Learning are underway in Avalanche[5]. Due to the variety of covered research settings, Sequoia has been used as the reference framework for the challenge held at the *2nd CLVision Workshop*[6] in which a Continual Reinforcement Learning track based on Monster Kong [168] was proposed.

Getting started example of Sequoia

```python
# Create the setting
setting = TaskIncrementalSLSetting(dataset="mnist")
# Create the method
method = BaseMethod(max_epochs=1)
# Apply the setting to the method to generate results.
results = setting.apply(method)
print(results.summary())
```

FIGURE 4.11: *Getting started* example of Sequoia, showing the creation and use of the Split-MNIST benchmark.

[5] https://github.com/ContinualAI/avalanche-rl
[6] https://sites.google.com/view/clvision2021

Chapter 5

Continual Learning Applications

Chapters 2 and 3 covered the theoretical aspects regarding benchmarks and techniques that can be used to model realistic scenarios and to learn when confronted with such complex benchmarks. However, apart from the previously described experimental results, we argue that the best way to show how the aforementioned achievements can be used is to propose practical applications that use these concepts.

In this chapter, the *CORe (C)ontinual (O)bject (Re)cognition* application is described, which demonstrates that AR1*-free with Latent Replay (Section 3.2) can be applied in practice on a smartphone (Section 5.1). In addition, we show that this idea can be used to enable continual learning capabilities in even more constrained devices, such as Ultra-Low-Power (ULP) boards (described in Section 5.2), which feature extremely limited computational and memory resources.

In addition, a practical application for Recycling Code[1] recognition is proposed in Section 5.3, in which incremental self-supervision capabilities are applied to a real application able to detect and identify recycling codes using real images captured using smartphones with varying degrees of image quality.

Finally, an overview of participated and organized competitions is given in Section 5.4, which describes in detail two competitions featuring realistic benchmarks and performance-based scoring mechanisms.

5.1 The CORe App

Efficiently learning at the edge is a challenging research problem for current machine learning research. In particular, the peace of advancements within the deep learning field has often been linked with a significant increase in computational and memory requirements. Deep models are often trained on remote servers and only later deployed with frozen learning capabilities on embedded devices. However, adapting prediction models at the edge is fundamental to preserve the private nature of users' data and to customize prediction models on the fly based on the specific user needs, even without an internet connection. On-device personalization could indeed provide more targeted and adaptive functionalities based on the user interaction and newly acquired data. However, on-device training is not only a matter of efficiency (i.e., memory and computation) but mostly about learning from shifting data distribution. Accumulating all the data seen over the lifetime of the deployed system and re-train the whole model from scratch on all the data becomes quickly impossible,

[1]Wikipedia entry on *Recycling codes*: `https://en.wikipedia.org/wiki/Recycling_codes`

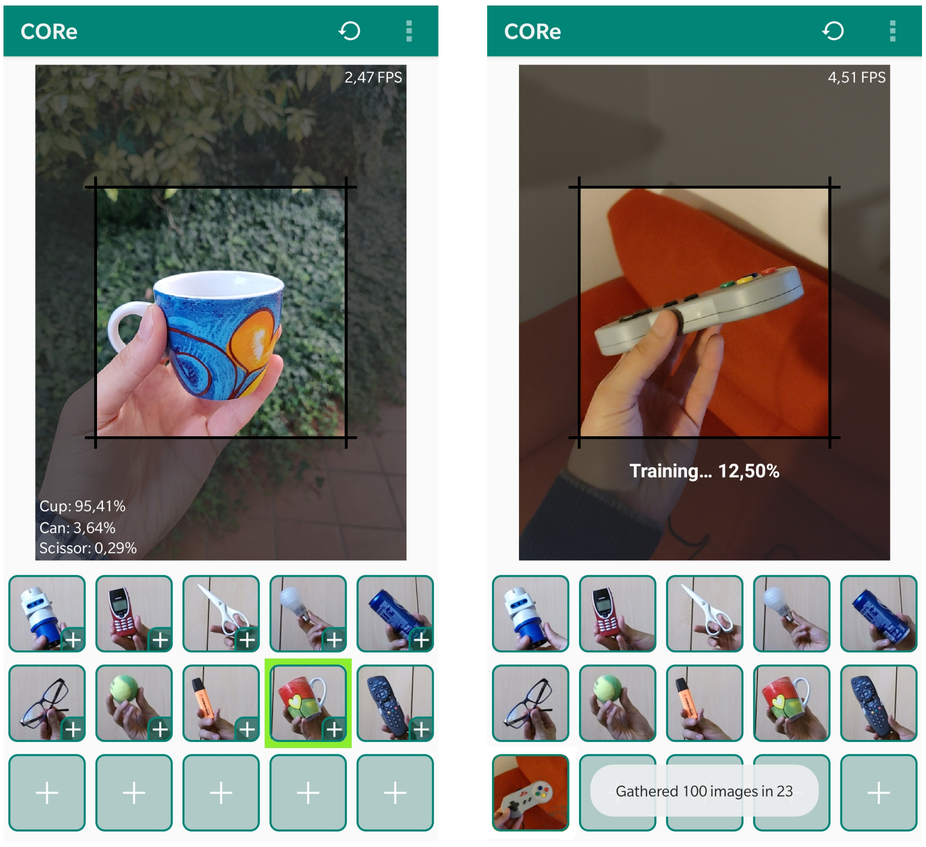

FIGURE 5.1: The CORe Application User Interface.

especially with frequent real-time updates. On the other hand, just updating the prediction model using only the newly available data incurs in catastrophic forgetting [107], and in many applications retaining previous knowledge is mandatory.

This section focuses on the design and deployment of the hybrid continual learning strategy *AR1** (described in Chapter 3) on a native Android application for real-time on-device personalization without forgetting. In Sections 3.1 and 3.2, AR1* with Latent Replay has been already shown to be efficient and flexible on natural video sequences constituting small non-i.i.d. batches of data. However, running that algorithm on edge devices is not straightforward as it poses several additional challenges. In particular, in this section we: i) detail the implementation of the Android application with native Java and C++ code based on Caffe framework (Section 5.1.1); ii) discuss how AR1* can be better parametrized in terms of Accuracy-Computation-Memory trade-off (Section 5.1.2); iii) introduce an extension of CORe50 for the life-after-deployment experiments running on CPU-only edge devices and we assess AR1* performances w.r.t. other state-of-the-art strategies (Section 5.1.2); iv) propose a two phases consolidation to achieve both real-time fast update and off-line delayed optimization (Section 5.1.1). The native Android application source code, along with the Avalanche-based [97] scripts used to reproduce the results are made available to further stimulate research in this area[2].

[2]https://github.com/lrzpellegrini/CL-CORe-App

5.1.1 CORe: an Android Application Based on Caffe

The (C)ontinual (O)bject (Re)cognition application is an Android application running on common smartphones[3]. The goal of CORe is to show how continual learning can solve a practical computer vision problem. The target scenario resembles a real lifelong learning setup where the user tries to improve the object recognition capability of its knowledge model; the user can be a robot, a static vision system, or even a person wearing camera-equipped smart glasses.

The application starts in inference mode (Figure 5.1, left): a continuous stream of frames is received from the rear camera and shown on the screen. In inference mode, the application continually shows the top-3 predicted categories for the object depicted in the central part of the image (highlighted by a surrounding greyed-out area). The application is packaged with a model pre-trained on the CORe50 dataset, which consists of video sessions of objects commonly found in domestic or office environments. CORe50 contains video sessions of 50 different objects belonging to 10 categories [95]. Considering that the goal of the application is a coarse-grained classification, the initial model here used was pre-trained on the categories instead of the objects. The user can switch to training mode by either selecting an existing category or an empty slot. By selecting an existing category the user can incrementally improve the recognition capability of the model for that category, which can be useful if an object is misclassified. A new category can be added by selecting one of the five empty slots in the last row. This triggers the image gathering phase, during which a 20 seconds video is taken, for a total of 100 frames. The training phase is then carried out using the AR1 algorithm with latent replay (Figure 5.1, right); in the current version of the application latent replay takes place at the penultimate layer of the model (*pool*), but alternative implementations are possible. In particular, a two stages consolidation can be envisaged where a quick (real-time) update takes place at the output layer and a slower (but more precise knowledge organization) is performed in the background affecting deeper layers. This is also in line with biological learning where the hippocampus can make recent knowledge immediately accessible while consolidation in the cortex is carried out throughout sleep-wake cycles [60]. The application is equipped with a MobileNetV1-1.0 in 32-bit floating-point format working with 128x128 input images [66] and leverages a fixed-size replay buffer of just 500 patterns for which only latent representations are kept. The application uses the latent replay mechanism to drastically reduce the training time: latent activations of the new images are computed while the frames are gathered so that at the end of acquisition the sole part of the model following the latent replay can be quickly trained with these precomputed activations. The performance/accuracy trade-off of this choice is discussed in detail in Section 5.1.2. It should be noted that using the penultimate layer as the latent replay layer, the memory occupation of a replay buffer of 500 patterns is less than 2MByte.

A custom version of the Caffe framework is used for both the inference and training phases. Other popular deep learning libraries for mobile devices (i.e., TensorFlow-Lite, PyTorch-Mobile) still do not support training operations as they only package the operators strictly required for the forward pass. Moreover, to operate with these libraries, the model has to be properly adapted and frozen. On the contrary, by compiling Caffe and its dependencies from scratch one can exploit complete deep learning functionalities. For comparison, the authors of [32] propose a similar Android application based on TensorFlow-Lite, which uses an experimental external

[3]CORe App video demo: `http://bit.ly/latent-replay`

TABLE 5.1: Final accuracy and on-device training time (in seconds). The *pool* variant allows for shorter training phases while sacrificing the accuracy on new categories. The *conv5_4* variant is a good trade-off between *pool* and *input*.

	AR1 variant:	**Pool**	**Conv5_4**	**Input**
Final accuracy	Initial categories	**97.54%**	97.16%	93.62%
	New categories	65.33%	73.39%	**79.07%**
Training time for a new experience on 100 new patterns, 8 epochs	Feature extraction[4]	8.93	6.70	-
	Forward pass	0.14	125.05	436.33
	Backward pass	0.04	4.82	4.81
	Weights update	0	37.21	58.88
	Overall	0.18	167.08	500.02

library to handle the training on the last fully connected layer. That library uses a TensorFlow-Lite model as a fixed feature extractor.

5.1.2 Experiments

A proper benchmark must be defined in order to assess the ability of the CORe application to learn new categories of objects. In Section 2.2.2, the complex NICv2-391 benchmark was proposed in which the system is required to continually learn from a stream of 391 experiences. In that benchmark, each experience can either depict an already encountered or a new object which makes it very akin to the use case covered by the CORe application. NICv2-391 cannot be used here because CORe50 already serves as the pretraining dataset used to populate the initial 10 categories. However, following the idea behind NICv2, here we propose an extension of the benchmark where new categories are encountered over time. Therefore, we collected an extended dataset containing 25 objects grouped in 5 new categories; 12 sessions (both indoor and outdoor) are available for each object: 9 are used for training while 3 are left for the test set. Therefore, the final benchmark consists of 225 experiences and it is modeled as a NIC scenario by following the NICv2 generation procedure introduced in Section 2.2. The dataset was captured by using different smartphones thus obtaining videos whose quality, field-of-view, and stability are different from those in CORe50. Our experiments are aimed at measuring the degree of forgetting on the original CORe50 categories in a complex scenario where no video sessions depicting objects of these categories are seen again.

In the proposed experiments, the AR1 strategy is compared against two pure-replay strategies namely "class balanced" and "unbalanced". In order to allow for a fair comparison, the same setup used in the application is considered: the fixed-size replay buffer (500 patterns) is initially filled with instances from the CORe50 dataset by selecting the same number of instances for each class. In all the compared strategies, the instances to be inserted and replaced are chosen randomly. In particular, the “class balanced” approach balances the number of instances contained in the replay buffer across categories while the “unbalanced” strategy does not employ any kind of balancing mechanism and replaces a fixed amount of patterns (10) after each training experience. For the replay strategies, the result of their *pool* variant is reported, in which the model layers below the last fully connected layer are kept frozen, thus using it as a feature extractor. For AR1, the *pool* and *conv5_4* variants identify the

[4]The feature extraction step is executed concurrently with the image gathering phase and does not affect the user experience.

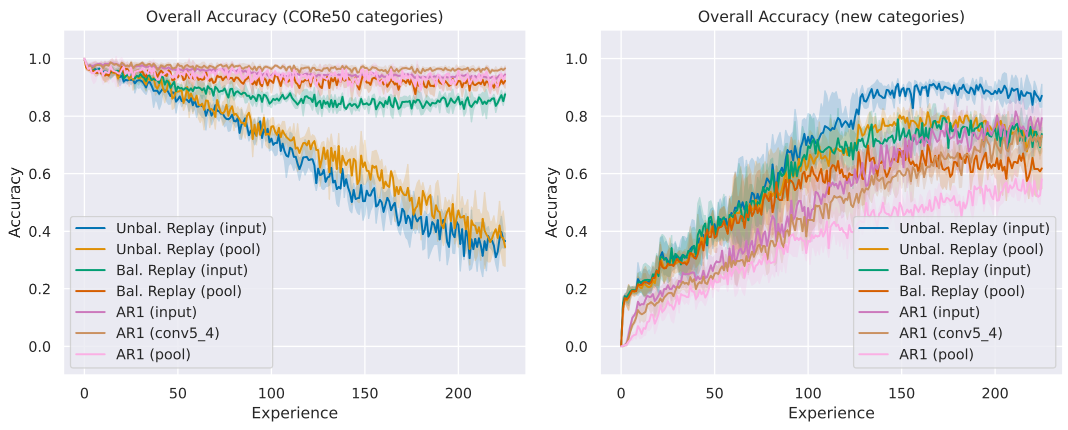

FIGURE 5.2: Test accuracy computed after each training experience for each strategy. Left: accuracy on the CORe50 categories; right: the accuracy on the new categories.

chosen latent replay layer. It is worth noting that the "unbalanced (pool)" strategy is the same one employed in the application described in [32]. For all strategies, the *input* variant identifies the replay from the input layer.

The experimental results reported in Figure 5.2 and Table 5.1 show that AR1 provides a good trade-off since it is able to learn the new categories without forgetting the old ones. The unbalanced replay mechanism shows good learning capabilities on the new categories but is not able to avoid forgetting on the CORe50 ones, on which the accuracy steadily decreases. Nevertheless, this minimal replay mechanism is still able to prevent an abrupt accuracy drop. The *pool* variant of the class balanced replay is able to retain the knowledge of the CORe50 categories but it does not reach the same accuracy performance of AR1 on the new categories. As expected, including the representation layers in the learning process improves the overall accuracy at the cost of increased computational overhead. The training phase of the chosen algorithm has to be completed in a certain time span based on the target user experience. The AR1 strategy allows for a fine-grained trade-off choice between accuracy and performance based on the selected latent replay layer. Table 5.1 shows a comparison of these different choices. Using *conv5_4/dw* as the latent replay layer allows for near-optimal knowledge conservation on the original CORe categories while demonstrating a good training trend on the new ones. However, in the current version of the application, we implemented the *pool* variant to maximize training efficiency. A further optimization could be achieved by combining *pool* and *conv5_4* according to the two steps consolidations discussed in Section 5.1.1.

5.1.3 Remarks on the CORe App

Learning continually at the edge may open the door to several privacy-preserving and personalized AI systems. However, on-device training is subject to many real-world constraints, and strict computational and memory limitations. In this section, we showed that a hybrid continual learning strategy, *AR1*, can provide an efficient and effective approach for sustainable on-device personalization while controlling the forgetting of previously acquired knowledge.

5.2 Continual Learning on Ultra-Low-Power devices

AI-powered edge devices currently lack the ability to adapt their embedded inference models to the ever-changing environment. To tackle this issue, Continual Learning (CL) strategies aim at incrementally improving the decision capabilities based on newly acquired data. In this section, after quantifying memory and computational requirements of CL algorithms, we define a novel HW/SW *extreme-edge* platform featuring a low power RISC-V octa-core cluster tailored for on-demand incremental learning over locally sensed data. The presented multi-core HW/SW architecture achieves a peak performance of 2.21 and 1.70 MAC/cycle, respectively, when running forward and backward steps of the gradient descent. We report the trade-off between memory footprint, latency, and accuracy for learning a new class with *Latent Replay* (Section 3.2 when targeting the image classification task on the NICv2-391 benchmark based on the CORe50 dataset (Section 2.2.2).

Thanks to the parallelism of the low-power cluster engine, our HW/SW platform is 25× faster than a typical *microcontroller unit (MCU)* device, on which CL is still impractical, and demonstrates an 11× gain in terms of energy consumption with respect to mobile-class solutions.

5.2.1 Background

Novel sensors and smart devices are equipped with *ultra-low-power* digital processing platforms that process raw data locally and extract high-level information by applying intelligent algorithms, such as Deep Neural Networks (DNNs). While the DNN inference capability has been already demonstrated on extreme-edge devices [190, 22, 112], the training of DNN models still relies on GPU-based machines. Once trained, the inference models are deployed on edge platforms tailored for prediction-only tasks. However, these inference models cannot adapt to the present environment, which may differ significantly from the training data statistics.

A way out of this rigid *train-on-cloud – deploy-at-edge* model is the use of Continual Learning algorithms, that aim at adapting a network model to a new class of sensor stimuli. This process is extremely challenging because of the *catastrophic forgetting* [107, 153]. Safeguarding previous knowledge is a significant concern. Regularization techniques and replay-free methodologies, i.e., ones not reusing the initial training data, can tackle this problem; alternatively, replay-based techniques safeguard previous knowledge with incremental training over a combination of previous and new data. Among these latter techniques, in Section 3.2 the Latent Replay has been presented, which can be used to effectively learn new object classes based on new samples and compressed old data.

In this section, we present a Hardware/Software platform design to run for the first time Continual Learning algorithms at the *extreme-edge* on a MicroController Unit (MCU)-class architecture. In contrast with current *ultra-low-power* MCUs that are mostly tailored for DNN inference, we propose a system architecture that can run incremental learning tasks on-demand by activating a RISC-V-based 8-core cluster subsystem.

The contributions of this section are:

- We evaluate the computational and memory requirements of a CL algorithm with latent replay on the CORe50 NICv2-391 benchmark with a MobileNetV1 [66] model.

- We define a HW/SW architecture for CL at the *extreme-edge* based on the *Parallel Ultra Low Power platform (PULP)* [144, 127], as well as a tensor tiling strategy necessary to fit within the limited memory.
- We benchmark forward and backward steps for CL on PULP, and evaluate their performance and efficiency together with the energy-accuracy trade-offs of a MobileNetV1 network that learns over the CORe50 dataset.

The presented HW/SW design delivers an average performance of 1.84 MAC/cycle during the learning task, thanks to an almost ideal speed-up of 7.79× gained by the parallelization over 8-cores with respect to a single-core implementation. Employing our platform, we demonstrate Continual Learning over the CORe50 dataset by coupling the processing engine with external memories for low-bandwidth operations. The learning of a new class can be achieved in 1.5 h by retraining only part of the network parameters with a memory need of 70 MB for RW operations and 200 MB to store old Latent Replay data permanently. This CL setting leads to an accuracy of 72.5%, which is only 5% lower than retraining all the layers, but on it is 3.2× faster. Moreover, the proposed solution demonstrates to be 25× faster than typical low-power MCUs, and, given the estimated power cost of ≈70 mW, 11× more energy-efficient when compared to mobile-class solutions.

5.2.2 Machine Learning on Low-Power devices

Edge and extreme-edge devices have constrained memory and computing resources that make the on-device training challenging. Federated Learning addresses this limitation by distributing the training tasks on multiple devices [108, 80]. The work presented by Xu et al. [185] studied the trade-offs between computation latency for a training step and devices frequency over six mobile-class processors while aggregating the results on the server side. Some devices feature a higher frequency and power consumption with respect to others (up to 64% higher) to balance the latency time among the different devices. Yet, the most efficient mobile-class platforms still present a power consumption in the range of few Watts. In contrast to them, we present an HW platform that allows training with a power budget below 100 mW.

From a platform-perspective, processing devices tailored for machine learning tasks feature a power consumption varying between a few mW up to hundreds of Watts [138].

We focus on flexible SW-programmable and low power devices distinguishing between 1-10 W (*edge* device) or <1 W (*extreme-edge* device) respectively. Among the edge devices, we report the TPUEdge [51], which is developed mainly for embedded inference applications, and the Qualcomm Snapdragon 845 that features a quad-core CPU, 4 Kryo 280 Gold coupled with an embedded GPU, and a dedicated DSP, namely the *AI engine*. This latter device features a power consumption up to 4.5 W, which is above the requirement for an extreme-edge device that we target in this section.

Moving to the *ultra-low-power* spectrum, the *extreme-edge* single-core MCUs, e.g., the STM32 MCU series [164], feature low power consumption and compliant with the requirements of battery-powered sensors. However, this comes at the cost of the

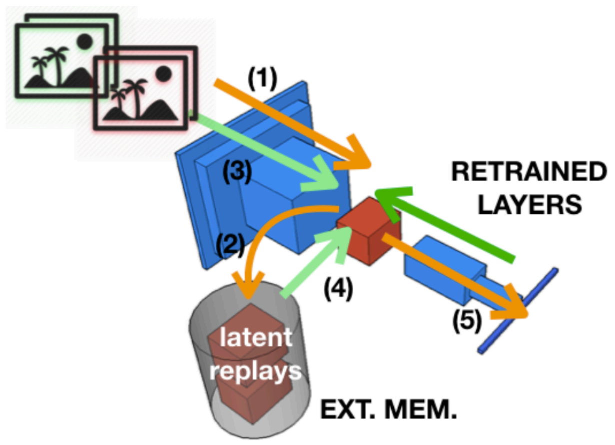

FIGURE 5.3: Latent Replay computational model. New images are elaborated up to the LR layer (1) and stored in the external memory (2). Afterwards, a mix of new images (3) and LRs (4) are used to retrain the final layers (5).

modest computational capacity, e.g. an STM32L4 device includes a single CPU running up to 48 MHz. To address this limitation, GAP8 [44] and MrWolf [127] platforms feature an MCU-based architecture accelerated by a multi-core cluster. In particular, MrWolf has a power budget as little as 150 mW when running the 8-core cluster at 450 MHz and also supports floating-point (FP32) arithmetic. Relying on the design principle of this latter systems, we present a HW/SW design to enable CL at the *extreme-edge*.

5.2.3 CL with Latent Replay

The goal of Continual learning techniques is to fine-tune deep networks based on new data samples, without retraining on the entire dataset. This section sketchs *i)* the computational model of the CL algorithm presented in Section 3.2 when targeting an extreme-edge platform and *ii)* compute its memory requirements.

Computational Model To bring CL with Latent Replay on a smart sensing platform, we model the incremental learning task to operate on a set of new data coming from a sensor, which is interfaced with an embedded digital processing engine.

Figures 3.4 and 5.3 illustrate the learning process that takes place over a set of data samples, including both N_I new images and N_{LR} old Latent Replay vectors. These latter are the activation feature maps of the Latent Replay (LR-th) layer, which are computed when feeding the network with a subset of training samples.

At runtime, a set of N_I new class data are sampled and converted into LR vectors. For this purpose, the network is fed with the new data up to the Latent Replay layer. The new activations are then mixed with the old LR tensors, and afterwards, the learning algorithm updates the parameters of the remaining layers. Hence, the dataset for a training step of CL is composed of old and new LR vectors. Typically, $N_I/N_{LR} = 1/5$ [122].

A more in depth description of AR1* and AR1*-free with Latent Rplay can be found in Sections 3.1 and 3.2.

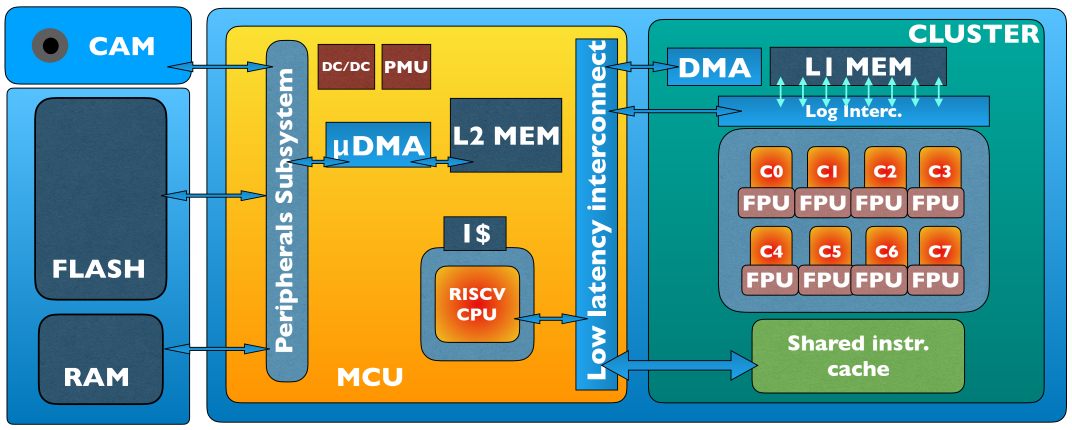

FIGURE 5.4: The proposed platform architecture for *extreme-edge* CL. The architecture consists of a cluster of 8 RISC-V tightly coupled cores featuring private FPUs and two shared L1 and L2 scratchpad memories.

Memory Requirements Consider a network model featuring N stacked layers. Besides the data for the learning process, a total of $N_w = \sum_{i=0}^{N-1} N_w(i)$ network parameters have to be stored in memory, where $N_w(i)$ represents the number of parameters at the i-th layer. Apart from the model weights, the following additional memory requirements are accounted for as follows:

- N_a is the total amount of intermediate features computed during the forward pass, which have to be preserved in memory for gradient computation;
- N_g is the number of gradient's components of the network parameters to be retrained.
- N_{Fi} is the number of parameters in the Fisher matrix, which is equal to the number of parameters to update.
- N_{fw} is the required memory footprint for temporarily storing activation feature maps during the forward pass. Its size is typically negligible with respect to other terms.

5.2.4 HW/SW platform for CL

In this section, we present our HW/SW platform design tailored for CL workloads. We evaluate the accuracy-latency-memory trade-offs based on the described system architecture.

HW Platform for edge learning

The proposed CL platform is based on the design principles of the PULP platform [46], which combines parallel programming for high-performance and ultra-low-power features.

The system architecture, which is depicted in Figure 5.4, is based on an MCU platform accelerated by a multi-core cluster of RISC-V cores. The MCU side features a single RISC-V core, namely the Fabric-Controller (FC), and a large set of peripherals. Besides the FC core, which is equipped with a private L1 data memory, the

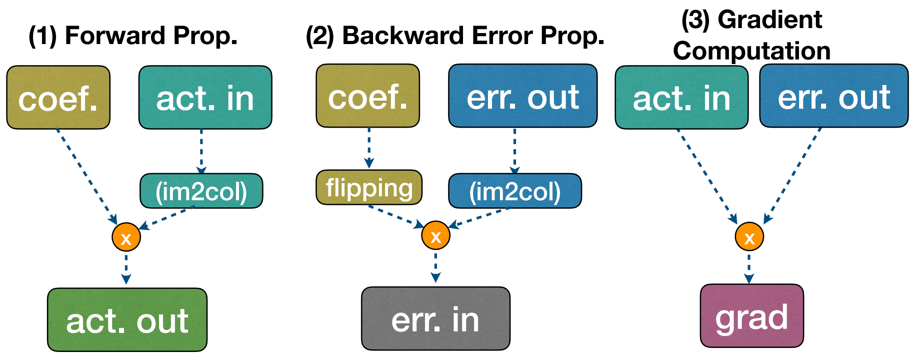

FIGURE 5.5: CL Computational Phases Graphs. (1) Forward step; (2) Error back-propagation; (3) Gradient Descent computation.

MCU-side of the platform includes a relatively large (256-1024 KB) on-chip L2 memory. The cluster features 8 processing elements (PE), each one including a RISC-V CPU equipped with a private Floating-Point Unit (FPU). To avoid memory coherency overheads and increasing area efficiency, the cores share a 64-256 KB L1 data scratchpad memory. Additionally, a multi-channel DMA engine handles data transfers between in-cluster and off-cluster memories.

The L2 memory is used as a temporary buffer for the peripheral data: a μDMA unit autonomously handles the data transfers between the L2 memory and the external peripherals. Among them, the system can be interfaced with a large L3 off-chip memory, through a parallel interface, and an external FLASH for data storage through Quad-SPI, e.g., a SD FLASH memory. Both external memories feature low power consumption when in idle state.

To efficiently implement power saving mechanisms, the FC core can switch-on the cluster on-demand at runtime, by controlling the internal cluster DC-DC regulator. Once powered-on, the FC core dispatches tasks on the 8-cores cluster, relying both on data- or task- parallelism for efficient and fast computation. On the other side, the system pays an energy overhead when activates the cluster computation on-demand. Likewise, external memories can be power-gated if not running the learning tasks.

SW stack for Continual Learning

In this section, we detail the mapping of the CL data flows on the architecture defined in Figure 5.4. Because of the high memory requirements and their constant nature, Latent Replay activations are kept in an external FLASH memory. During the learning phase, activations are loaded into the L2 memory to be replayed. On the other side, an external L3 DRAM memory is used to store network parameters, gradients, and intermediate activations values. Data transfers from L3 RAM to L2 occurs in the background of the computation thanks to the pipelined μDMA operation.

CL updates the model's parameters based on the back-propagation algorithm, which consists of the basic operations depicted in Figure 5.5. During the *Forward pass* of a network, the computational layers apply kernel convolutions over the activation values (*act_in*). Each convolution is reshaped as a 32-bit Floating-Point (FP32) generic matrix multiplication (GEMM) by means of an *im2col* transformation applied on the input activation tensor (*act_in*); the *im2col* tensor is stored in a shared L2 buffer.

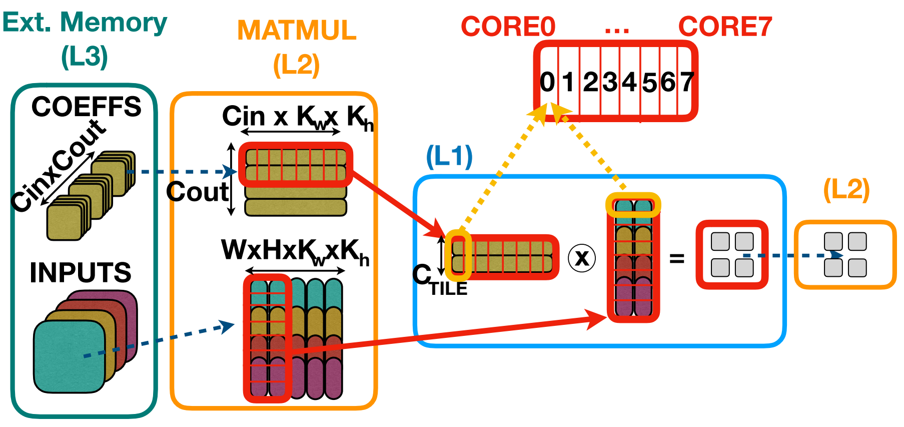

FIGURE 5.6: GEMM dataflow and parallelization scheme. GEMM uses tiling to benefit from L2 and L1 data locality. Parallelism is achieved using data-parallelism (SPMD) to distribute the work to the 8 cores.

The results of the forward kernels, i.e., the intermediate activation tensors, are then stored back in L3 memory, since they are going to be used to compute the network update step. During the *Backward Pass*, which runs up to the *LR*-th layer, the gradient w.r.t. the activations (*err_out*) is propagated to the next layer, after applying a convolution (GEMM) operation with the flipped (*coeff*) vector (graph (2) in Fig. 5.5). Likewise, the computation of the gradients of the parameters (*grad*) are computed using GEMM convolution kernels, which takes as inputs the *act_in*, computed during the forward pass, and the *err_out* tensors.

Figure 5.6 represents the memory management and the parallelization scheme to implement the computational graph of Figure 5.5 in the target platform. Data (i.e., gradients, coefficients, activation values, or input data), which resides on the L3 memory, is loaded into the on-chip L2 and L1 memories for layer-wise processing. Given the on-chip memory limitations, for some layers, tensor operands have to be sliced, and computation is performed on sub-tensors. This tensor slicing is called *tiling* and previous works already presented a lightweight strategy, hence implementing a software-cache mechanism for DNN deployment [16]. In Fig. 5.6 we visualize the tiling of the coefficients along the output-feature dimension C_{out}: a subset of C_{TILE} filters, hence a total of $C_{TILE} \times C_{in} \times K_w \times K_h$ parameters, is transferred to the cluster L1 memory for the computation. Inside the cluster, the FP32 GEMM is *parallelized* over 8 cores, both for forward and backward passes, according to a data-parallelism paradigm. Such scheme holds for the types of computations reported in Fig.5.5. The operand *INPUTS* of Figure 5.6 refers either to the *act_in* or the *err_out* tensors. During forward propagation and gradient computation, the parallelization operates over the input-feature dimension C_{in}, whereas during backward error propagation the workload is parallelized over the C_{out} dimension of the activations tensor.

5.2.5 Experimental Results

In this section, we evaluate our HW/SW design over the NICv2-391 benchmark, and we compare our solution against other proposed CL algorithm implementations.

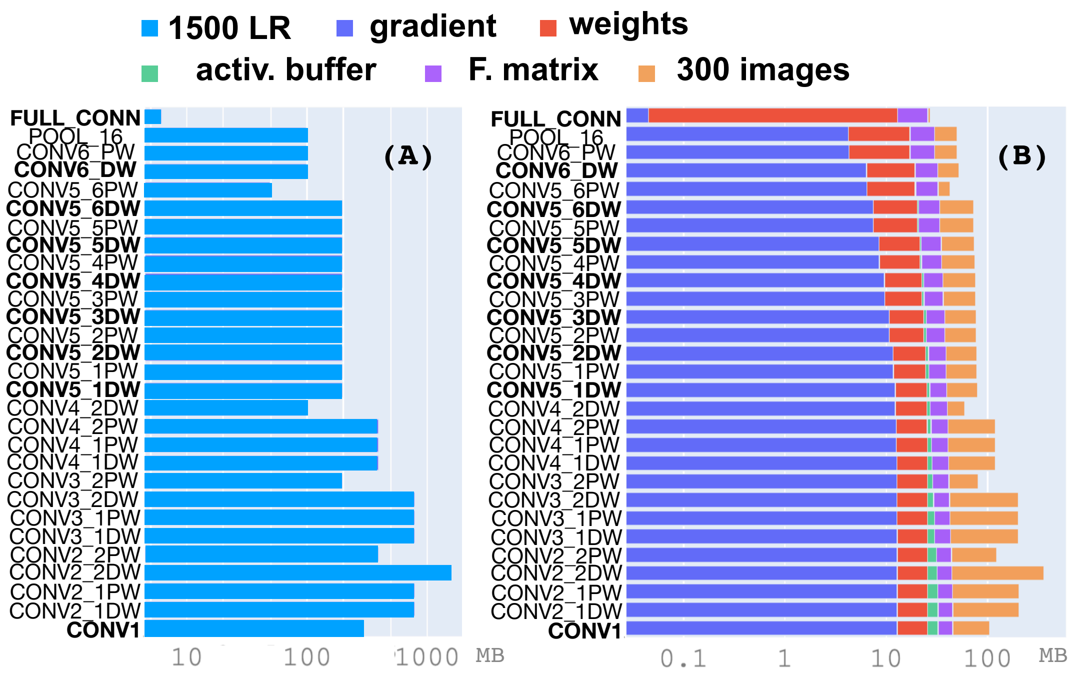

FIGURE 5.7: FLASH and RAM Memory Footprint for different LR cuts.

Experimental Setup

CL Task and Dataset To benchmark our HW/SW design, we consider a CL task consisting of learning new object classes from the CORe50 dataset [96]. For a more in-depth description of the dataset, please refer to Chapter 2. To make the benchmark close to a real application, new objects are discovered sporadically over time. Thus, the three-way NICv2 protocol introduced in Section 2.2.2 is used, where the first insertion of the samples of a new class is balanced over the training batches (see figures in Sections 2.2.2 and A.2). In particular, in NICv2-391, each of the 390 incremental batches includes only one training session (300 images) of a single class.

We rely on the experimental settings described in Section 3.2, where a MobileNetV1 model with input resolution of 128x128 and width multiplier 1 is used. In this section, we indicate the individual layers of the networks with the same naming convention used in that section and also reported in Figure 5.8. The model is initially trained to distinguish only 10 of the 50 classes of the dataset. Each incremental learning step applies the gradient descent algorithm over a single batch composed of 1500 LRs vectors (30 LRs for every class) and 300 new images. This learning step iterates for 8 epochs.

Evaluation Metrics Computation latency is measured through a cycle-accurate simulator of the proposed architecture. The base version of the simulator, which we extended for this study, is validated against *Register Transfer Level (RTL)* for the open-source PULP platform and MrWolf, that is our reference *System on a Chip (SoC)*. The selected running frequency is 150 MHz. The reported accuracy represents the overall precision reached by the network after complete learning on the CORe50 NICv2-391 benchmark.

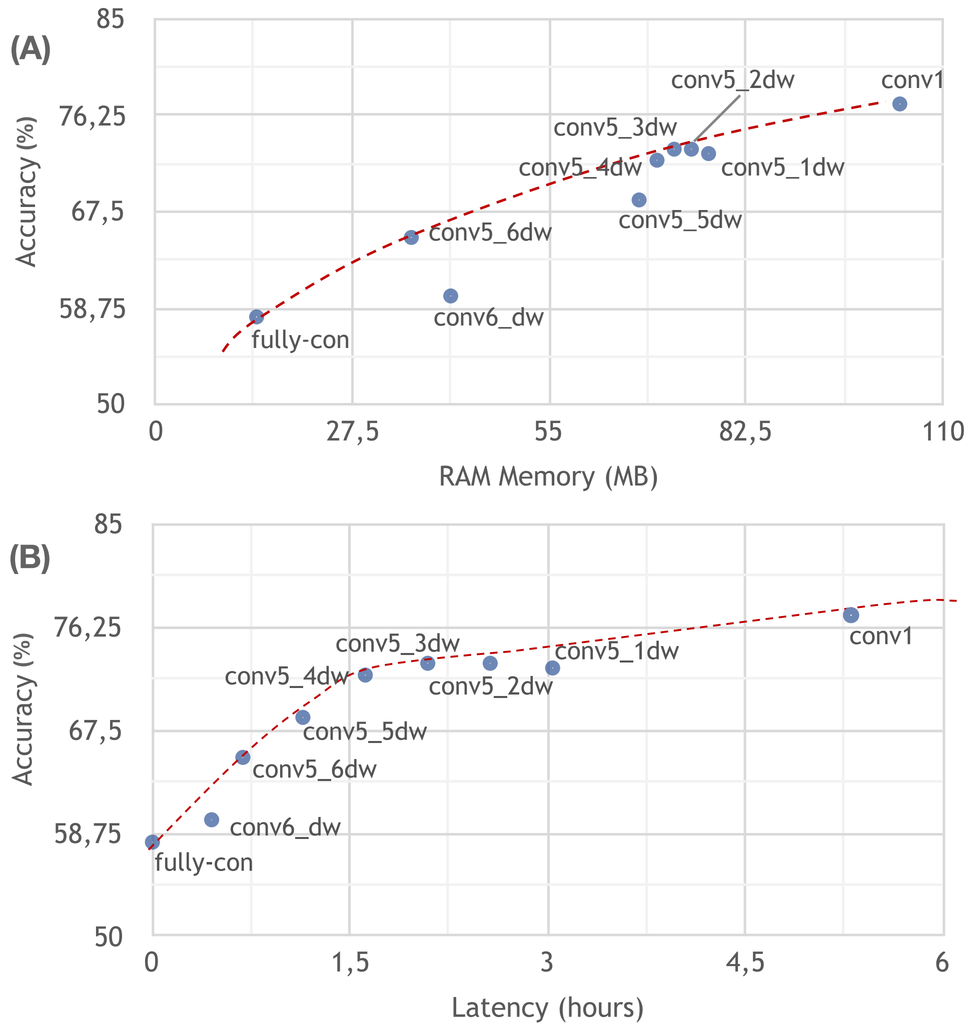

FIGURE 5.8: Latency-Memory-Accuracy trade-off for different LR cuts. The red dashed lines represent the two Pareto sets. The red dot (*conv1*, the first layer) assumes that we retrain the whole network.

Memory Evaluation

Figure 5.7 shows the memory footprint needed (A) to permanently store the 1500 Latent Replay activations, i.e. the ROM memory requirement, and (B) to store the new data samples, the network parameters and gradients, the intermediate features maps for the backward pass and the approximated Fisher matrix components. All these values result in a memory footprint of few MBs and change over time hence they are stored on the RAM. Note that the memory requirements vary depending on the chosen *LR* layer: retraining only the last layer (indicated as mid_fc7) requires the storage of smaller LR vectors, gradients, and activation feature maps than selecting any other middle layer as the LR layer (e.g. $conv5\_4/dw$). This also explains the higher memory demands when choosing a LR layer closer to the first one. Concerning the non-volatile FLASH memory (Figure 5.7(A)), the requirement ranges from few MB (6 MB if the last layer is the LR layer) to up to 300 MB, when retraining the full network based on the image samples (the only CL setting not featuring LRs). A typical-size SPI FLASH memory is therefore sufficient in all cases. Also note that LR arrays are accessed sporadically only to perform retraining. Hence, the external FLASH memory can idle, or even be power-gated, for the rest of the time.

Figure 5.7(B), instead, plots the RAM requirements, which also includes the 300 LRs related to the new images that require >60% of the overall memory space. The memory breakdown shows that individual memory terms vary depending on the LR layer selection. Only the size of network parameters is constant. Figure 5.8(A) combines the RAM memory requirements with the network accuracy, which were demonstrated to be state-of-the-art on the CORe50 dataset in Section 3.2. If setting the LR layer as one of the last layers, the accuracy drop increases because of the higher number of frozen parameters. If imposing a constraint of 32 MB for the external DRAM memory, the only match is the retraining of only the last layer (*mid_fc7*). But in this case, accuracy is limited to 58%, which is nearly 20% lower than the baseline (retraining all the layers). Conversely, to reach a higher precision of 72.2% (*conv5_4/dw*), 70 MB of external RAM is needed, which is achieved using a multi-bank DRAM memory.

Latency-Accuracy Trade-off

Figure 5.9 reports the latency measured on our HW/SW platform over various computation kernels, either for single-core or 8-core execution. The performance is expressed as MAC/cycle in the case of forward and backward passes for three representative layers of our benchmark network, namely Pointwise, Depthwise, and Fully Connected layers. Peak values of performance are achieved in Pointwise layers, reaching 2.21 MAC/cycle in forward and 1.70 MAC/cycle in backward computations. The performance measured for the forward pass results is higher than backward because GEMM operates on larger vectors, hence the computation density is higher. The speed-up achieved by the 8-cores implementation against single-core reaches 7.79× on average, close to the theoretical limit of 8×.

Figure 5.8(B) reports the accuracy-latency trade-off on the learning task, with a cluster clock frequency of 150 MHz. The latency refers to the total time to learn a new class, which depends on the chosen Latent Replay layer. To reduce the memory access time overhead, we consider a tiling mechanism between L1 and L2 memories, realized using the DMA engine to minimize latency overhead (below 5% [16] with respect to the execution latency when on all data reside in L1). The L3 to L2 overhead instead is almost negligible because data transfers occur concurrently to the computation on large sub-tensors. Retraining the whole network with 1500 replays results in an accuracy of 77.3%, and the overall latency for 1-class learning is 318 min. A faster solution is obtained by choosing LR=*conv5_4*. In this setting, the learning latency for a new class is 98 minutes and the reached overall accuracy is 72.2%. If we reduce the number of network layers to retrain, the accuracy drops more substantially, as shown in Fig. 5.8. Among the faster configurations, the *ultra-fast* solution retrains only the Fully Connected layer, featuring a computation latency of 867 ms to learn a single class, but with a low 58% accuracy which sets a lower bound for the CL algorithm. Our proposed platform opens up the possibility to perform CL on extreme-edge nodes, which is not practical on current-generation commercial MCUs. For example, compared with a state-of-the-art low-power STM32L476 running at 48 MHz, our platform runs the learning task 25× faster.

Energy Estimation

We evaluate the energy consumption based on the CORe mobile application presented in Section 5.1, which operates over training batches composed of 500 LRs and 100 new frames belonging to a previously unseen class. The mobile-class device

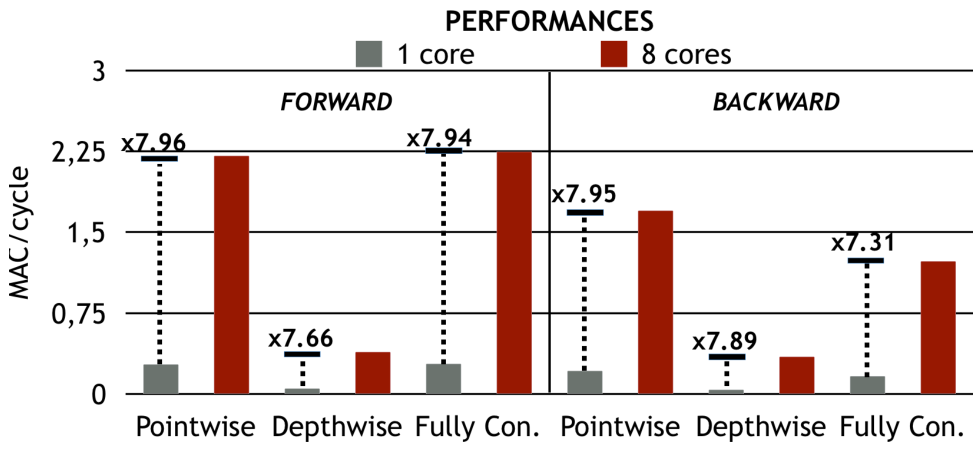

FIGURE 5.9: Measured Forward and Backward Throughput and parallelization efficiency.

used for the reference design of the CORe app is a OnePlus6 featuring a Qualcomm Snapdragon 845 processor. To estimate the energy consumption of the proposed HW/SW platform, we refer to power measurements from the silicon prototype of MrWolf [127], which operates at 9 MMAC/s/mW and features an average power of 70 mW when running at 150 MHz.

Given an application scenario requesting 1 inference/sec and 1 retraining step per hour, our platform features a total energy consumption of 34.2 J per hour in the fastest CL configuration setting (LR=*mid_fc7*). This implies a battery lifetime of about 710 hours (we consider a 3100 mAh battery), which is 11× higher than a Snapdragon845-based solution (assuming 0 Watt idle power). To gain higher accuracy, hence retraining more layers, i.e., LR=*conv5_4* layer, the energy consumption increases to 1530 J per hour, therefore leading to 15.8 h of battery life.

5.2.6 Remarks on CL on ULP devices

In this section, a novel HW/SW architecture specifically tailored for the execution of CL algorithms was presented. In particular we assessed the memory and computational requirements of a replay-based CL algorithm with Latent Replay. Moreover, we showed the accuracy-latency trade-off on the proposed *extreme-edge* system, which results to be 11× more energy-efficient than previous mobile-class processors. The achieved results motivate to further explore Continual Learning on extreme-edge devices.

5.3 Weak self-supervision for Recycling Codes recognition

Waste sorting at the household level is a virtuous process that can greatly increase material recycling and boost circular economy. Unfortunately, the large number of recycling codes printed on products makes this process unfriendly for many users. In this section, we propose a vision-based mobile application to support users in recognizing recycling codes for proper waste sorting. Our experimental results show that the combination of deep-learning techniques, image processing pipelines, and weakly supervised iterative training schemes (based on domain knowledge), allows for the development of an effective application with minimum effort in terms of data collection and labeling, which is one of the main obstacles toward the successful application of deep-learning techniques to real-world problems.

In particular, the proposed approach consists of an incremental training procedure that uses weak feedback coming from the users to self-boost the performance of the symbol detector. In the future, more extensive use of incremental and continual learning approaches may allow for more tailored learning, thus enabling user-specific conditions to be accounted for (image quality, recurrent symbols, etcetera).

5.3.1 Background

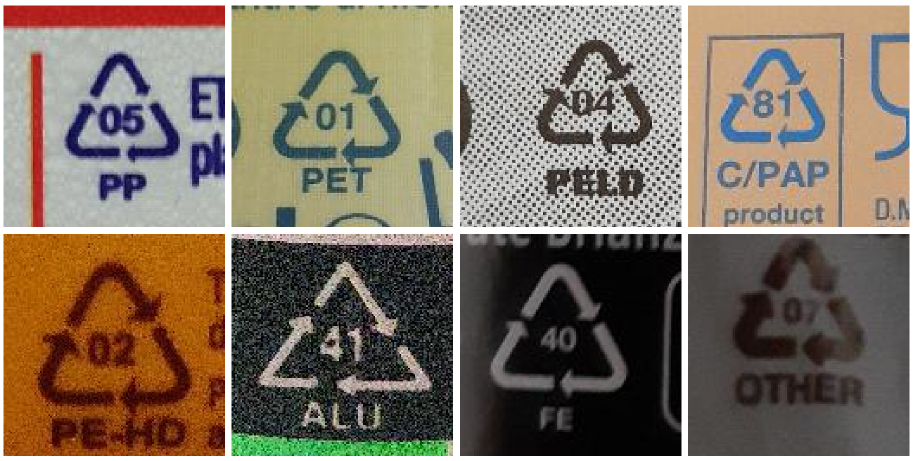

FIGURE 5.10: Some examples of recycling symbols.

Recycling codes are used to identify the materials from which products are made, to facilitate their recycling. Although the symbols used to depict recycling codes are not many and their shape can be easily identified (e.g., chasing arrows in Figure 5.10), the total number of codes/materials is high[5] and some of them are rarely used. Hence, users cannot easily remember the codes and the corresponding recycling rules. The aim of this work is the development of a mobile application that can support the users in recognizing recycling codes for proper waste sorting. Ideally, an application could directly recognize the different products from their visual aspect without relying on the recycling code; however, due to the huge amount of existing products and packages, their aspect variations, and the continuous launch of new items, inferring waste categories from images is very challenging. For this reason, most of the approaches proposed in the literature [187, 109, 12] operate in restricted settings (e.g., the limited number of materials, uniform background, etc). Furthermore, in some cases, the visual discrimination of materials is not possible, and even humans need to rely on the associated recycling symbols.

In the proposed approach, as a user points his smartphone camera to the zone where the code is printed, the application must promptly recognize the symbol and display recycling instructions. The development of such an application is not trivial because: (i) the symbols and the associated labels (number/text) are usually small in size and properly framing them with a smartphone camera is critical because of the close distance necessary for the required zoom and the risk of capturing out-of-focus or

[5]See https://en.wikipedia.org/wiki/Recycling_codes

blurred images; (ii) taking a single shot and sending it to a remote server for recognition requires a stable internet connection and would often lead to rejecting the chosen sample thus requiring multiple attempts. Performing online recognition from the camera video stream is better, but it requires on-device processing at a proper frame rate.

As of today, the most obvious approach seems to be selecting a state-of-the-art Convolutional Neural Network (CNN) which is efficient enough for real-time inference on a smartphone and training it as a classifier on a sufficiently wide labeled dataset. This approach has some drawbacks:

- recycling codes are numerous and a dataset containing a sufficient number of examples per class is not available. Collecting from scratch such a dataset is time-consuming and finding examples of some "rare" classes can be an issue not easy to address with data augmentation techniques;
- while on-screen feedback can guide the user in properly framing the symbols (see Figure 5.11), it is not realistic to impose that the searched pattern is well centered and size normalized. Therefore, we must move from a classification to a detection task with the consequence that: i) the model complexity increases; ii) the symbol bounding boxes need to be labeled in the training dataset.

To solve the aforementioned critical issues, we propose a two phases approach that combines a deep learning architecture with an image processing pipeline. The main contributions of this work are:

- the combination of deep models with image processing pipelines (based on domain knowledge) to overcome the limitations of data-driven approaches when training data are limited and does not cover all the cases;
- the definition and validation of a weakly supervised iterative training scheme. While this is demonstrated in the context of recycling code recognition, the approach is general enough to be applied to other real-world scenarios;
- the demonstration that recycling code recognition can be performed on-device with good accuracy; even if some applications for recycling code recognition already exist (see [155]) we are not aware of accompanying scientific papers or technical descriptions and therefore we cannot make a direct comparison with the proposed approach. However, we provide an ablation study where the most important contributions of our proposal are isolated and benchmarked against baseline methods;
- we release the novel dataset used in our experiments. The dataset is composed of short video sessions gathered by users depicting recycling codes in various conditions.

5.3.2 The approach

A graphical scheme of the proposed approach is depicted in Figure 5.13. An efficient classification CNN (MobileNet v2 [149]) is extended with a regression head for the prediction of the symbol position (bounding box). This model is much simpler and more efficient than a Yolo [136] or SSD [94] since we assume that a single object of interest is present in the image. The model is trained to localize a recycling symbol (chasing arrows) in a single image: the model output is the confidence on the

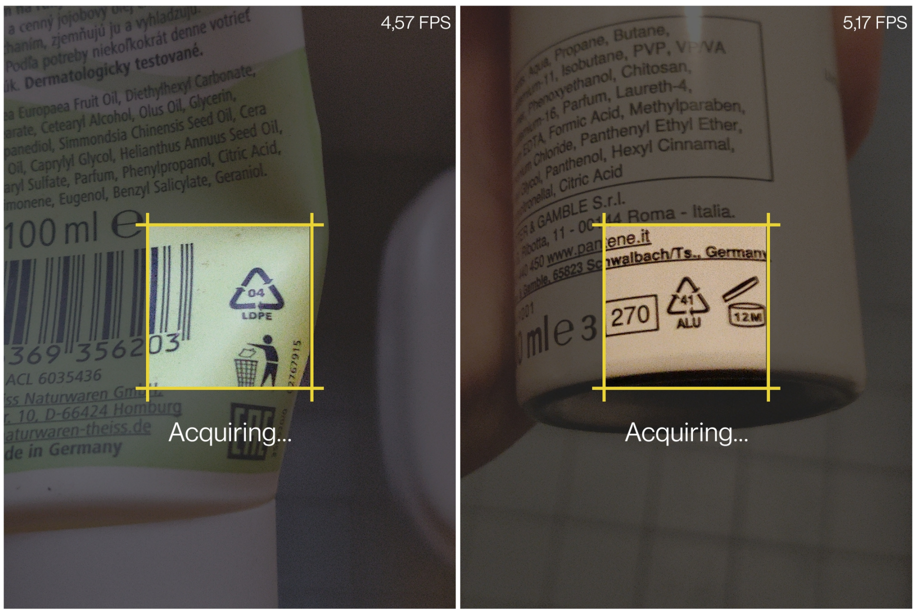

FIGURE 5.11: The acquisition interface of the proposed application. The user is asked to frame the symbol inside the central area (with a yellow border). The external greyed-out area is ignored during recognition but is useful to help the user center the symbol.

symbol presence (classification) and the bounding box coordinates (regression). It is worth noting that recycling codes of the different classes (associated with different numbers and texts) are not discerned at this level, and therefore the classifier makes a binary decision (symbol found vs no-symbol found). Extending the application to work with multiple recycling symbols (e.g., circle, trashcan, etc.) is not complex and some hints are provided in the following sections. The frames where a symbol is found are passed to a second processing phase where an OCR algorithm is used to extract the numbers and text identifying the recycling category.

Although the above two phases decomposition allows training the system without a comprehensive dataset (in terms of real-world combinations of symbols, numbers, and text), some problems persist.

In fact, the text region detectors and OCR algorithms tested, including state-of-the-art OCR on-the-wild [163, 10], are often disturbed by the presence of graphics (e.g. lines, drawings) in the proximity of the text and their accuracy significantly degrades when the text is not well isolated from the background (see Figure 5.12). To solve this problem, instead of retraining an OCR to work in the specific context of our application[6], we introduced an image post-processing pipeline that, starting from the bounding box returned by the model, identifies and refines the text regions to be processed by a general-purpose OCR.

Furthermore, training a regression head requires a sufficient number of bounding box labeled examples in the training dataset, whose markup is a boring and time-consuming task. To overcome this problem we propose a weakly supervised (iterative) training where:

[6]Again, this would require a representative training dataset, which is difficult to collect.

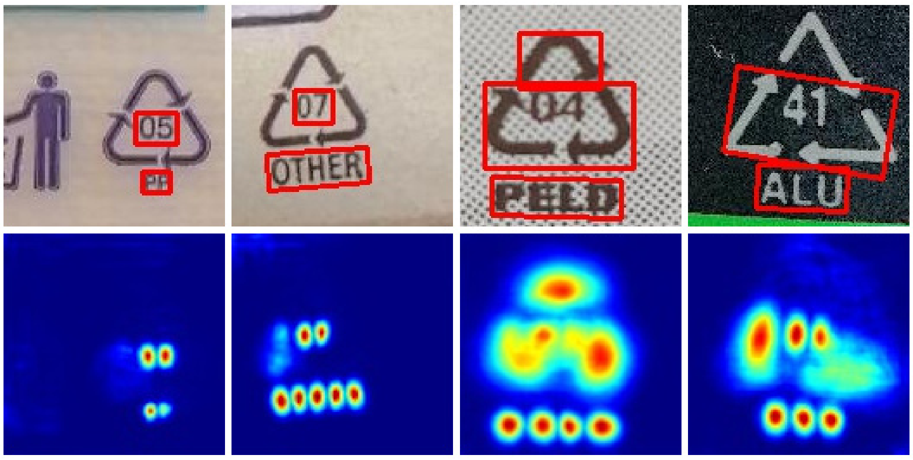

FIGURE 5.12: Some results obtained by applying the Tesseract OCR [163] to the text regions returned by the CRAFT text detector [10]. In the first row, the predicted text positions are highlighted in red and the OCR results are in blue. The second row shows the heatmap used by the text detector. The number and text in the first two columns are properly extracted, while the OCR failed in the last two columns because of poor text region detection.

- the model is initially trained on a small recycling code dataset: the SevenPlastics dataset available in Kaggle [7];
- the initial model, deployed in a beta version of the app, has been used to collect many new examples that are labeled (both the class and the bounding box) with self-supervision and weak user engagement;
- the model is then (iteratively) trained on the union of the initial data (full supervision) and the successive examples (weak supervision), achieving a relevant accuracy improvement. The weak supervision comes from end-users in the form of the class label (the symbol depicted in the frames, without the bounding box). The goal of the iterative training procedure is to tune the detector to improve symbol localization.

5.3.3 Phase I: frame classification and bounding box regression

The model used in Phase I is a MobileNet v2 (128x128 input, depth multiplier = 1.0) pre-trained on ImageNet [33], where:

- the output classification layer is replaced with a new layer with just 2 classes (simbol vs no-symbols).
- a new fully connected layer with 4 neurons is attached to the second last layer to be trained as a regressor to predict the symbol position (bounding box).

This model is trained on a subset of the Kaggle recycling code dataset consisting of 454 frames whose bounding boxes have been manually marked by us. An equal number of negative examples (i.e., images belonging to the "no-symbol" class) are added by randomly cropping ImageNet [33] samples[8]. The training is carried out for 100 epochs with the Adam optimizer. Actually, this training session is a tuning for all

[7]https://www.kaggle.com/piaoya/plastic-recycling-codes

[8]Since Kaggle images are cropped around the symbol we cannot use out-of-center portions to create negative examples.

the levels except the two output heads whose weights are randomly initialized. As shown in Section 5.3.6, even a small dataset of recycling codes allows the model to reach a good binary classification accuracy. However, the symbol locations returned by the regressor are often imprecise and do not allow to accurately crop the text regions.

5.3.4 Phase II: box refinements and OCR

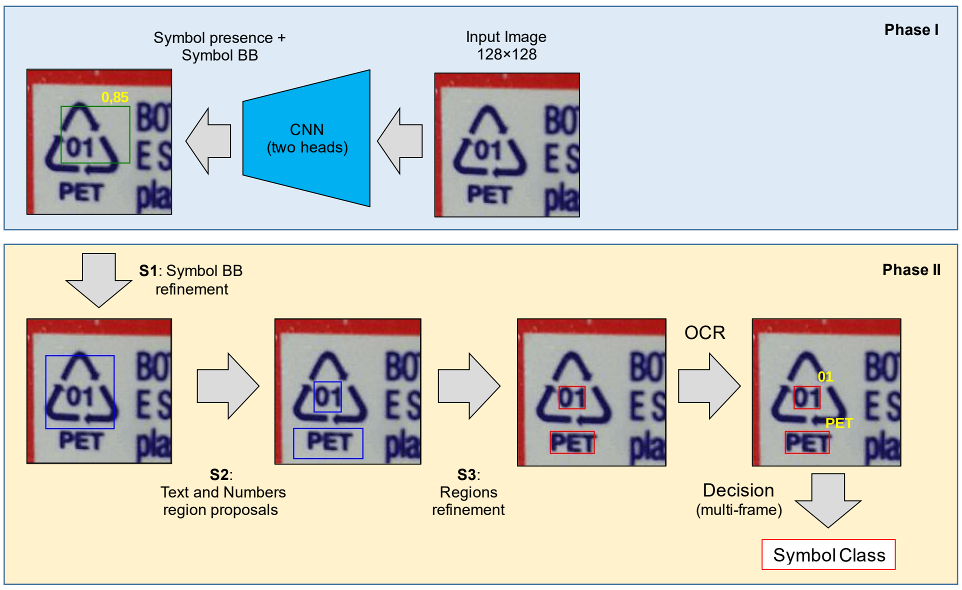

FIGURE 5.13: Overall schema of the proposed approach.

Figure 5.13 highlights the sequence of processing steps performed for the classification of a single frame. The CNN model of Phase I outputs a confidence score on the presence of a recycling symbol (0.85 in the example) and a bounding box denoting his position in the frame. If the confidence is lower than a threshold the frame is immediately discarded, otherwise the bounding box position is refined (step S1). The regions where we can expect to find numbers and text can be then hypothesized (step S2) based on domain knowledge (i.e., the geometric shape of the symbols and the average position/size of its textural attributes). In the case of the chasing arrows symbol, the internal region should contain a number and the base region a text: however, this is not always the case and more than two region proposals can be passed-on if necessary. Again, since the region proposals are not accurate enough for reliable OCR, a refinement step (S3) is necessary for segmentation. Each refined region is then processed through Tesseract OCR and the results are passed to a multi-frame decision-maker for final classification.

Hereafter we provide further details on steps S1, S2, S3, and the final OCR and multi-frame decision. Intermediate results are graphically shown in Figure 5.14. Further examples are reported in Figure 5.15. Since in this paper we focus on a chasing arrow symbol, the image processing steps are tailored to this symbol. However, we believe that simple variants of proposed pipelines can be designed for other common recycle symbols. The parameters of the processing steps have been tuned by extracting geometric statistics from the Kaggle labeled dataset.

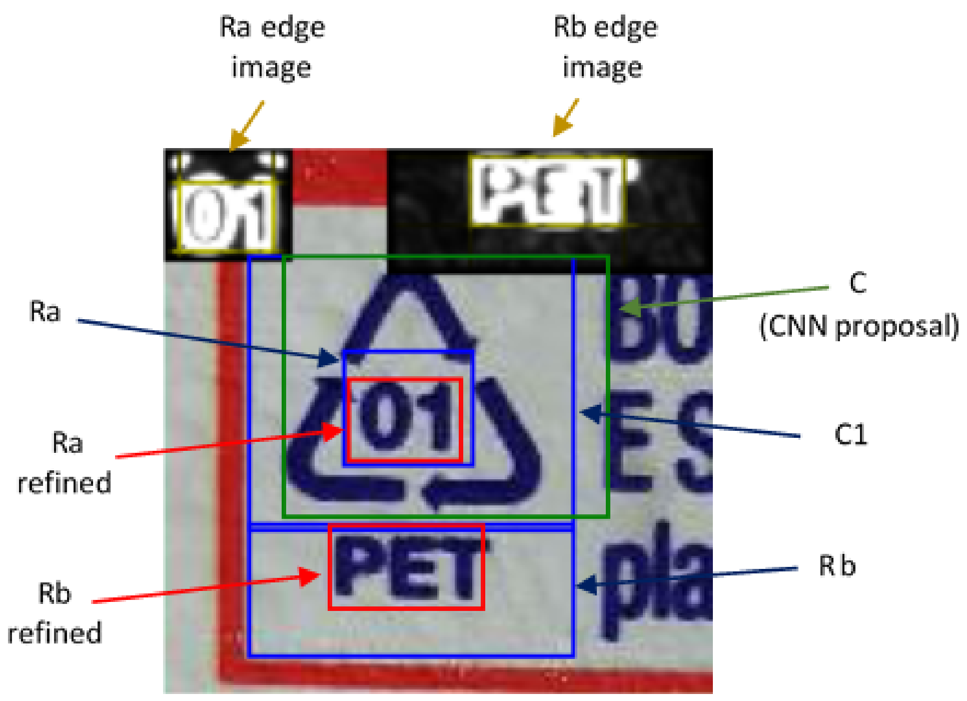

FIGURE 5.14: Graphical visualization of intermediate processing steps on an example image.

S1: Symbol bounding box refinement

The rectangle returned by the CNN is used to crop a subimage $C = [x_{left}, y_{top}, x_{right}, y_{bottom}]$, which is first converted to grayscale and then processed as follows.

- Compute the vertical axis of symmetry x_{sym} of C by minimizing the mean pixel intensity abs-difference between two small regions at the right and left of each hypothesized vertical axis. Since the considered symbol is nearly symmetric (with respect to y) this simple step allows a better horizontal centering.
- Select a bottom region B as $[x_{sym} - w/2, y_{bottom} - o, x_{sym} + w/2, y_{bottom} + o]$ where w is the initial crop width, $(w = x_{right} - x_{left})$ and o is a 20 pixel offset.
- Compute the magnitude of horizontal edges (abs of Sobel vertical filter 3x3) in B and apply a Gaussian blur (3x3) to reduce noise.
- Integrate the edge magnitude along the rows in B and select y_{new_bottom} as the row with highest magnitude. The rationale behind this step is that the chaining arrows symbol has a triangular shape and the triangle base produces strong horizontal edges.
- The region

  $C1 = [x_{sym} - w/2, y_{top}, x_{sym} + w/2, y_{new\_bottom} + o2]$

  is returned, where $o2$ is a small constant offset (6 pixels).

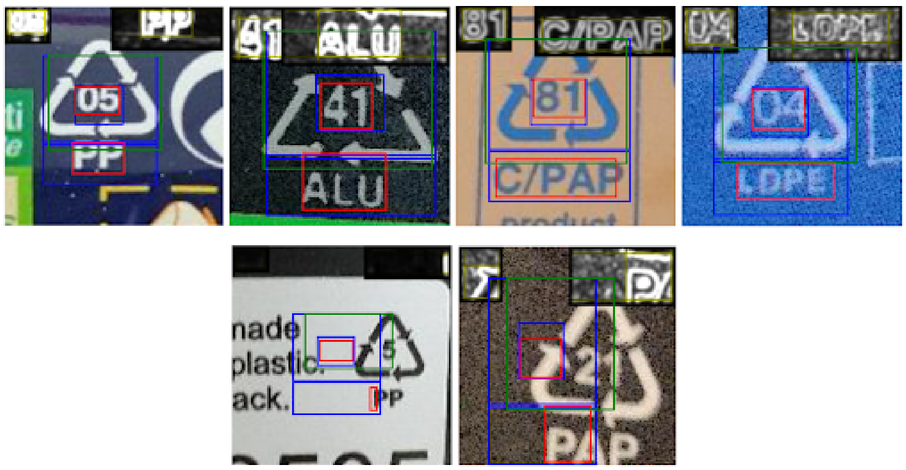

FIGURE 5.15: The first row shows some examples where the proposed processing is successful. In the second row some errors are shown; errors are typically due to a bad initial estimation of C by the CNN.

S2: Region proposals

Two region proposals Ra and Rb can be defined at fixed positions relative to C1. Ra is coarsely centered in the triangle center and its (width, height) are proportional to the triangle sides. Rb is positioned below the triangle base and is as large as the triangle. Due to the empirical nature of the above rules, Ra and Rb are enlarged (with an extra offset) to avoid losing portions of the searched regions. Even if Ra should contain a number (material code) and Rb a text (material acronym), this is not always the case and a more flexible decision is necessary after OCR. Furthermore, for increased robustness more than two proposals can be generated in this step, since the multi-frame decision adopted is general enough to tolerate this.

S3: Regions refinement

Ra and Rb subimages are converted to grayscale and each of them is shrinked by moving its 4 sides toward the center in order to precisely frame the text:

- Compute the magnitude of the gradient (Sobel filters 3x3) in B and apply a Gaussian blur (3x3) to reduce noise. Because of the enlargement in the previous step we expect an external offset region where the gradient magnitude is small.
- Integrate the magnitude over rows. Iteratively move the region top(bottom) side toward the center until the integrated magnitude over the current row is higher than a threshold.
- Integrate the magnitude over columns. Iteratively move the region left(right) side toward the center until the integrated magnitude over the current column is higher than a threshold.

OCR and multi-frame decision

The refined Ra and Rb are passed to Tesseract OCR [163], version 4.1.1[9]. Their grayscale intensities can be optionally inverted: in fact, we noted that Tesseract

[9]Tesseract 4.1.1: https://github.com/tesseract-ocr/tesseract/releases/tag/4.1.1

works better with dark text on bright backgrounds. To decide whether to apply the inversion, the average value of pixel intensity in an external boundary region (likely belonging to the background) is compared with the average intensity of the remaining pixels.

Tesseract can be configured to work with a restricted list of allowed characters. The OCR process is executed the first time by allowing only digits [0...9]. If a valid numerical symbol code is not returned, then a second attempt is made by passing a set of alphabetical characters obtained as the union of the characters used in all the symbol acronyms. Considering that number and text may appear in all proposed crops, the OCR engine is run a maximum of 4 times (2 on the refined Ra and 2 on the refined Rb) for each frame.

Even if the designed two-phase approach greatly improves single frame classification (see Table 5.2) some frames remain critical mainly due to: out-of-focus, symbol too close/far from the camera, excessive rotation or skew, and flares. Therefore a multi-frame fusion, based on the sum rule and a fixed threshold, is adopted to improve the overall accuracy. The pseudocode is reported in Algorithm 1. It is worth noting that a symbol receives a score of 0.5 if either the number or the text matches, and a full score (1.0) is assigned only in case of a simultaneous match. The threshold used in our experiments is 1.5: this means that 2 fully matching frames need to be associated with a given symbol before recognition or, alternatively, a higher number of partially matching frames is necessary.

The scoring mechanism could be further improved by assigning lower scores (but not zero) when a partial match is detected: for instance, a score of 0.25 could be assigned to “HDPE” if the OCR returns “HOPE” which is not in the dictionary. A normalized Levenshtein distance [89] can be used to grade scores. A further enhancement could be achieved by adopting an OCR working with a dictionary at word levels instead of at character levels.

5.3.5 Iterative training

As explained in Section 5.3.3, the CNN (classifier + regressor) is first trained on a subsample of the Kaggle SevenPlastic dataset (hereafter DB0), obtaining the model M0 which is embedded in the first version of the application. A small group of users was involved in a beta testing stage. When they note that the pointed symbol code is correctly classified no further action is required; otherwise, they provide the correct symbol code through a simplified graphical interface[10]; in the latter case, the video frames of the current session are stored in the device and later transferred to a remote workstation. At the end of the beta testing stage, we collected a total of 129 video sessions corresponding to error cases. While the ground truth code labels are available, the bounding boxes are missing and our goal is to avoid their tedious and expensive manual labeling. Therefore, for each frame of each session, we used model M0 to estimate the symbol position C and applied the steps S1, S2, S3, and the OCR (multistep fusion is not necessary here). If after OCR both the number and text are coherent with the ground truth symbol code we assume that the refined box C1 (produced by S1) can be reliably used as a ground truth bounding box: therefore, we add this frame to a new dataset P1. A new training DB1 dataset can be then assembled as the union of DB0, P1, and N1, where N1 is an extra set

[10]The application allows to select the symbol from a list and to associate it to all the frames in the current video.

```
Input: A sequence of frames $F_i$ in the current recognition session; $D$: A
       dictionary of recognizable symbols; model: the CNN model with
       classification and regression heads; $S$: the scores of elements in $D$.
       Initialized to 0.
for for each frame $X$ in $F_i$ do
    has_symbol, C = model(X)
    if not has_symbol then skip frame
    C1 = S1(X, C)
    Ra, Rb = S2(C1)
    $Ra_{refined}$, $Rb_{refined}$ = S3(Ra, Rb)
    $sym_{ra}$ = OCR($Ra_{refined}$)
    $sym_{rb}$ = OCR($Rb_{refined}$)
    if $sym_{ra}$ in $D$ then $S[sym_{ra}]$ += 0.5
    if $sym_{rb}$ in $D$ then then $S[sym_{rb}]$ += 0.5
    if (exists a $sym_f$ such that $S[sym_f] > threshold$) and ($S[sym_f] > S[sym_i]$ for
     each $sym_i$ != $sym_f$) then
        return $sym_f$
    end
end
return "unknown"
```

Algorithm 1: Pseudo-code of the proposed approach. D is a many-to-many dictionary that considers equivalences such as different numerical symbols tied to a common text label (and viceversa). For instance, "PAP" is linked to numerical codes {20, 21, 22} while numerical code 02 is linked to text codes {"PEHD", "HDPE"}. The score fusion mechanism keeps these equivalences in consideration (omitted for simplicity in the pseudo-code).

of negative examples (with the same size of P1) randomly sampled from ImageNet. Finally, a new model M1 can be obtained by retraining the CNN on DB1. The whole procedure can be iterated more times to improve the application accuracy. As the application accuracy increases with the iterations, the amount of user supervision is progressively reduced.

5.3.6 Experimental results

As explained in Section 5.3.5, the proposed approach has been trained incrementally. The initial training dataset DB0 is composed of 454 frames extracted from the Kaggle SevenPlastic dataset. Bounding boxes have been manually provided for DB0 and T0: for each instance, we marked three bounding boxes corresponding to the ground truth position of C, Ra, and Rb. Both DB0 and T0 have been extended with an equal number of negative examples by randomly cropping ImageNet data.

The test set T0 used across all the evaluations has been initially collected by a group of 9 users (each with her/his smartphone). T0 consists of 93 videos including 4519 frames and 14 different recycling symbols. The symbol class and the bounding boxes (C, Ra, and Rb) have been manually marked.

At training iteration 1, DB1 is the union of DB0 with P1 and N1 where: (i) P1 contains 1699 frames extracted from 36 videos collected by beta users; as explained in Section 5.3.5 only the frames properly classified are selected and their symbol bounding boxes have been self-generated; (ii) N1 is a set of negative examples of the same size of P1.

Finally, for training iteration 2, DB2 is obtained without further user involvement, but using model M1 to reclassify P1 samples. This resulted in an increased number of frames selected (from 896 to 1001) and more accurate self-generation of the ground truth bounding boxes.

Table 5.2 reports the results obtained after training the system on DB0, DB1 and DB2. Accuracy is reported for single frames or videos (session); as expected, the multi-frame fusion boosts the accuracy. In general, the proposed weakly supervised iterative training proved to be very effective: at the frame level, the classification accuracy increased from 55.5% to 71.7% to 73.14% and, at video level from 83.9% to 89.2% to 92.5%.

The symbol binary classification of the CNN (i.e., the accuracy of) also significantly increased after training on DB1, because of the new (disjoint) set of symbols introduced. Analogous improvements can be noted in the intersection over union (IoU) metrics of the bounding boxes.

	Training Iter: 0 **Train DB: DB0** **Test DB: T0**	**Training Iter: 1** **Train DB : DB1** **Test DB: T0**	**Training Iter: 2** **Train: DB2** **Test DB: T0**
Video classification accuracy	83.9% 78/93 - correct 5/93 - unknown 10/93 - wrong	89.2% 83/93 - correct 3/93 - unknown 7/93 - wrong	92.5% 86/93 - correct 2/93 - unknown 5/93 - wrong
Frame classification accuracy	55.52%	71.74%	73.14%
Symbol binary classification (CNN)	86.91%	99.76%	99.68%
C - IoU (CNN)	62.38%	80.02%	84.06%
C1 - IoU	68.45%	82.38%	84.14%
Ra refined and Rb refined (IoU)	Upper box: 56.92% Lower box: 63.22% Average: 58.68%	Upper box: 66.18% Lower box: 70.84% Average: 67.16%	Upper box: 64.39% Lower box: 71.37% Average: 66.30%

TABLE 5.2: Accuracy results over training iterations.

Visual error analysis revealed that the 7 videos that were not correctly classified after the second training iteration (see Figure 5.16) are characterized by: the lack of the numerical codes, the presence of textual codes inside the symbol far from the expected position, and very low quality (e.g. out of focus) images. Providing multiple proposals for both the text and number regions could help in solving many of these cases.

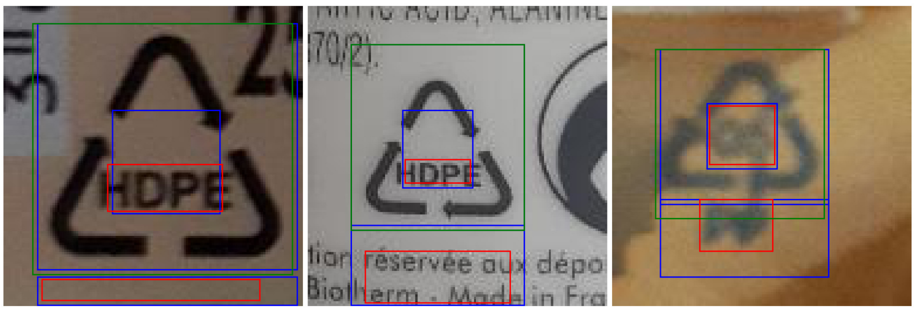

FIGURE 5.16: Examples of residual errors after the second training iteration. In the first and second columns, a textural code is printed inside the symbol instead of a number, and its width extends beyond the maximum expected one, so the refined region Ra (red rectangle) cannot include the text border. In the last example, the image is too blurred.

In table 5.3, we report the time required by the different stages/steps measured on a smartphone S10+ running Android 11. The most time demanding step is OCR; however, this step is performed only when a symbol is detected and the overall user experience remains quite fluid.

Step name	**Average time per image (ms)**
Phase I: Image classification and symbol detection (CNN inference)	24.35
Phase II: S1	1.09
Phase II: S3 (both boxes)	0.48
Phase II: OCR (both boxes)	143.49

TABLE 5.3: Time required by the different phases/steps, obtained as average on 10 examples.

Ablation study

To understand the contribution of the domain-knowledge steps we performed an ablation study consisting of two further experiments:

1. in the former we removed steps S1 (symbol bounding box refinement) and S3 (text regions bounding boxes refinement). In this case, Ra and Rb are obtained by S2 directly applied to C. Results in Table 5.4 prove that S1 and S3 have a very relevant role. For example the frame classification accuracy drops from 55.5% to 24.3%;

2. in the latter we focused on training iteration 1 where for the examples introduced in P1 the ground truth of the symbol bounding box is given by C (and not C1); furthermore P1 contains all the negative examples collected during beta testing and not only those whose text and number are validated after OCR (see Section 5.3.5). We denote this scheme as basic self-supervision. Results in Table 5.5 show that the exploitation of domain knowledge leads to improved accuracy.

	Training Iter: 0 Without S1 and S3	**Training Iter: 0 Full system**
Video classificationaccuracy	72.0% 67/93 - correct 10/93 - unknown 16/93 - wrong	83.9% 78/93 - correct 5/93 - unknown 10/93 - wrong
Frame classification accuracy	24.39%	55.52%
Ra and Rb - IoU	Upper box: 25.56% Lower box: 27.59% Average: 26.59%	Upper box: 56.92% Lower box: 63.22% Average: 58.68%

TABLE 5.4: Accuracy after training iteration 0, in the case of a full system (column 2) and removal of S1 and S3 (column 1).

	Training Iter: 1 Basic self-supervision	**Training Iter: 1 Full System**
Video classification accuracy	87.1% 81/93 - correct 6/93 - unknown 6/93 - wrong	89.2% 83/93 - correct 3/93 - unknown 7/93 - wrong
Frame classification accuracy	66.56%	71.74%
C - IoU (CNN)	74.33%	80.02%
C - IoU	78.40%	82.38%
Ra and Rb - IoU	Upper box: 60.95% Lower box: 67.13% Average: 62.45%	Upper box: 66.18% Lower box: 70.84% Average: 67.16%

TABLE 5.5: Accuracy after training iteration 1, in the case of a full system (column 2) and a basic self-supervision approach (column 1).

5.3.7 Conclusions

Final users' engagement in waste sorting constitutes a fundamental asset towards a greener and more sustainable processes for material recycling. However, product heterogeneity and varying regulations (e.g. depending on geographical location) may pose serious questions about the feasibility and accuracy levels of such classification. While the complete automation of waste sorting is far from being possible in industrial and particularly in household settings, in this section we proposed a vision-based mobile application that may help final users in the waste sorting process by automatically recognizing product recycling classification with high degrees of accuracy.

The developed methodology, based on the simple idea of focusing on recycling codes rather than object visual aspects, blends together state-of-the-art deep learning detection techniques as well as ordinary image processing pipelines and tools. This approach reflects a well-known recent trend in mixing machine learning models with specific domain knowledge in order to maximize task performance and reduce the overall amount of labeled data needed by the system. Furthermore, such a solution has been designed to run efficiently on embedded systems (e.g. ordinary smartphones) with highly constrained memory and computational budgets.

Finally, we proposed an interactive training scheme to address realistic scenarios where end-users actually contribute to the overall system predictions improvement only through weak supervision. We believe similar continual learning procedures may allow in the future not only for the incremental improvements in the recognition of currently considered recycling codes but may also result in the efficient adaptation to new recycling codes being introduced over time. Adaptation capabilities may be even leveraged for "personalized learning", with the simple yet effective idea of customizing prediction models to personal, user-specific conditions (e.g. camera settings, more frequent products to recycle, etc.) which may result in additional performance gains.

5.4 Continual Learning Competitions

The goal of a competition is to stimulate the research community to produce new and more effective solutions in promising research directions. For a field that has seen a sudden increase in popularity in the last years, challenges are very important events that can be used to bring the research community together and stimulate discussions. Competitions are usually held in the context of workshops. If new editions of a workshop are proposed across several years, the organizers may consider running a different challenge each year. For instance, challenges hosted at CLVision workshops (hosted in the context of the annual CVPR conference) featured novel elements of complexity that the participants had to overcome: the challenge hosted in the 1st CLVision workshop proposed a difficult benchmark made of nearly 400 incremental experiences, while the challenge hosted in the 2nd CLVision workshop proposed a complex continual reinforcement learning benchmark.

In this section, two challenges are detailed: the *IROS 2019-Lifelong Robotic Vision Competition* and the *1st CLVision Workshop Challenge at CVPR 2020*. Each challenge is born with different ideas in mind, but in both these challenges, a realistic *video dataset* (OpenLORIS [128] and CORe50 [95]) has been proposed. In addition, they both feature an *instance-based* classification problem. Another interesting point is that in both competitions, to compute the score of submissions, some performance-related metrics (training and/or inference time, model size, amount of replay instances used, etcetera) were taken into consideration. We argue that considering performance metrics may be essential to prevent distortions in the design of continual learning strategies submitted as solutions. Common distortions include the use of overabundant replay instances, the use of extremely time-expensive computations, and the use of excessively big models.

In the context of the IROS 2019 challenge, we participated as team "Unibo" by proposing an algorithm based on the union of LwF [92] and Latent Replay (Section 3.2). Our solution was able to reach the best score in terms of performance (final model size and inference time) and 2nd place overall. In the context of the 1st CLVision challenge, I took part in the technical organization of the efforts involved in the development of the *devkit* and the *evaluation of the finalists' solutions*.

5.4.1 The IROS 2019 competition

Humans have the remarkable ability to learn continuously from the external environment and the inner experience. One of the grand goals of robots is also building an artificial "lifelong learning" agent that can shape a cultivated understanding of the world from the current scene and their previous knowledge via an autonomous lifelong development. It is challenging for the robot learning process to retain earlier knowledge when they encounter new tasks or information. Recent advances in computer vision and deep learning methods have been very impressive due to large-scale datasets, such as ImageNet [33] and COCO [93]. However, robotic vision poses unique new challenges for applying visual algorithms developed from these computer vision datasets because they implicitly assume a fixed set of categories and time-invariant task distributions [41]. Semantic concepts change dynamically over time [157, 156, 159]. Thus, sizeable robotic vision datasets collected from real-time changing environments for accelerating the research and evaluation of robotic vision algorithms are crucial. For bridging the gap between robotic vision and stationary computer vision fields, we utilize a real robot mounted with multiple-high-resolution sensors (e.g., monocular/RGB-D from RealSense D435i, dual fisheye images from RealSense T265, LiDAR,, see Figure 5.17) to actively collect the data from the real-world objects in several kinds of typical scenarios, like homes, offices,campus, and malls.

This section summarizes IROS 2019-Lifelong Robotic Vision Competition (Lifelong Object Recognition challenge) with dataset, rules, methods and results from the top 8 finalists (out of over 150 teams). Individual reports, dataset information, rules, and released source codes can be found at the competition homepage.

In particular, this section will describe in detail the main aspects of the challenge, our proposed solution (which achieved second place), and the winners' solution, while additional details (such as other winners' solutions) are reported in the appendix (Chapter D). For more details, we recommend that you refer to the official report [9].

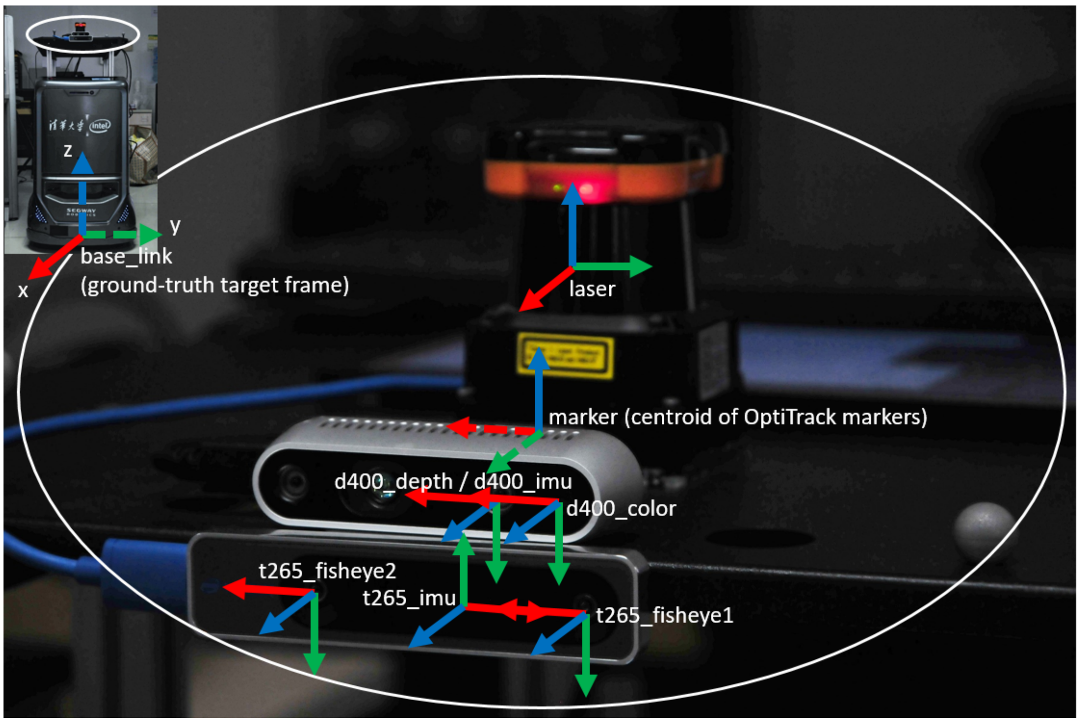

FIGURE 5.17: OpenLORIS robotic platform (left) mounted with multiple sensors (right). In OpenLORIS-Object dataset, the RGB-D data is collected from the depth camera.

Lifelong Robotic Vision - Object Recognition Challenge

This challenge aimed to explore how to leverage the knowledge learned from previous tasks that could generalize to new task effectively, and also how to efficiently memorize of previously learned tasks. The work pathed the way for robots to behave like humans in terms of knowledge transfer, association, and combination capabilities.

To our best knowledge, the provided lifelong object recognition dataset OpenLORIS-Object-v1 [158] is the first one that explicitly indicates the task difficulty under the incremental setting, which is able to foster the lifelong/continual/incremental learning in a supervised/semi-supervised manner. Different from previous instance/class-incremental task, the difficulty-incremental learning is to test the model's capability over continuous learning when faced with multiple environmental factors, such as illumination, occlusion, camera-object distances/angles, clutter, and context information in both low and high dynamic scenes.

OpenLORIS-Object Dataset IROS 2019 competition provided the 1st version of OpenLORIS-Object dataset for the participants. Note that our dataset has been updated with twice the size in content available at the project homepage with detailed information, visualization, downloading instructions and benchmarks on SOTA lifelong learning methods [158].

We included the common challenges that the robot is usually faced with, such as illumination, occlusion, camera-object distance, etc. Furthermore, we explicitly decompose these factors from real-life environments and have quantified their difficulty levels. In summary, to better understand which characteristics of robotic data negatively influence the results of the lifelong object recognition, we independently consider: 1) illumination, 2) occlusion, 3) object size, 4) camera-object distance, 5) camera-object angle, and 6) clutter.

1). **Illumination**. The illumination can vary significantly across time, e.g., day and night. We repeat the data collection under weak, normal, and strong lighting conditions, respectively. The task becomes challenging with lights to be very weak.

2). **Occlusion**. Occlusion happens when a part of an object is hidden by other objects, or only a portion of the object is visible in the field of view. Occlusion significantly increases the difficulty for recognition.

3). **Object size**. Small-size objects make the task challenging, like dry batteries or glue sticks.

4). **Camera-object distance**. It affects actual pixels of the objects in the image.

5). **Camera-object angle**. The angles between the cameras and objects affect the attributes detected from the object.

6). **Clutter**. The presence of other objects in the vicinity of the considered object may interfere with the classification task.

The version of OpenLORIS-Object for this competition is a collection of 69 instances, including 19 categories daily necessities objects under 7 scenes. For each instance, a 17 seconds video (at 30 fps) has been recorded with a depth camera delivering 500 RGB-D frames (with 260 distinguishable object views picked and provided in the

Level	*Illumination*	*Occlusion (percentage)*	*Object Size (pixels)*
1	Strong	0%	$> 200 \times 200$
2	Normal	25%	$30 \times 30 - 200 \times 200$
3	Weak	50%	$< 30 \times 30$

Level	*Clutter*	*Context*	*# Classes*	*# Instances*
1	Simple	Home/office/mall	19	69
2	Normal			
3	Complex			

TABLE 5.6: Details of each 3 levels for 4 real-life robotic vision challenges.

dataset). 4 environmental factors, each has 3 level changes, are considered explicitly, including illumination variants during recording, occlusion percentage of the objects, object pixel size in each frame, and the clutter of the scene. Note that the variables of 3) object size and 4) camera-object distance are combined together because in the real-world scenarios, it is hard to distinguish the effects of these two factors brought to the actual data collected from the mobile robots, but we can identify their joint effects on the actual pixel sizes of the objects in the frames roughly. The variable 5) is considered as different recorded views of the objects. The defined three difficulty levels for each factor are shown in Table. 5.6 (totally we have 12 levels w.r.t. the environment factors across all instances). The levels 1, 2, and 3 are ranked with increasing difficulties.

For each instance at each level, we provided 260 samples, both have RGB and depth images. Thus, the total images provided is around 2 (RGB and depth) $\times$260 (samples per instance)$\times$69 (instances) $\times$4 (factors per level) $\times$3 (difficulty levels) = 430,560 images. Also, we have provided bounding boxes and masks for each RGB image with Labelme [146]. The size of images under illumination, occlusion and object pixel size factors is 424$\times$240 pixels, and the size of images under object pixel size factor are 424$\times$240, 320$\times$180, 1280$\times$720 pixels (for 3 difficulty levels). Picked samples have been shown in Figure 5.18.

Challenge Phases and Evaluation Rules We held 2 phases for the challenge. The preliminary contest we provided 9 batches of datasets which contain different factors and difficulty levels, for each batch, we have train/validation/test data splits. The core of this incremental learning setting is, we need the first train on the first batch of the dataset, and then 2^{nd} batch, 3^{rd} batch, until the 9^{th} batch, and then use the final model to obtain the test accuracy of all encounter tasks (batches). The training/validation datasets can only be accessed during the model optimizations. We held the evluation platform on Codalab. There had been over over 150 participants during the preliminary contest and we chose 8 teams with higher testing accurries over all testing batches as our finalists.

For the final round, different from standard computer vision challenge [33, 93], not only the overall accuracy on all tasks was evaluated but also the model efficiency, including model size, memory cost, and replay size (the number of old task samples used for learning new tasks, smaller is better) were considered. Meanwhile, instead of directly asking the participants to submit the prediction results on the test dataset as standard deep learning challenges [33, 93], the organizers received either

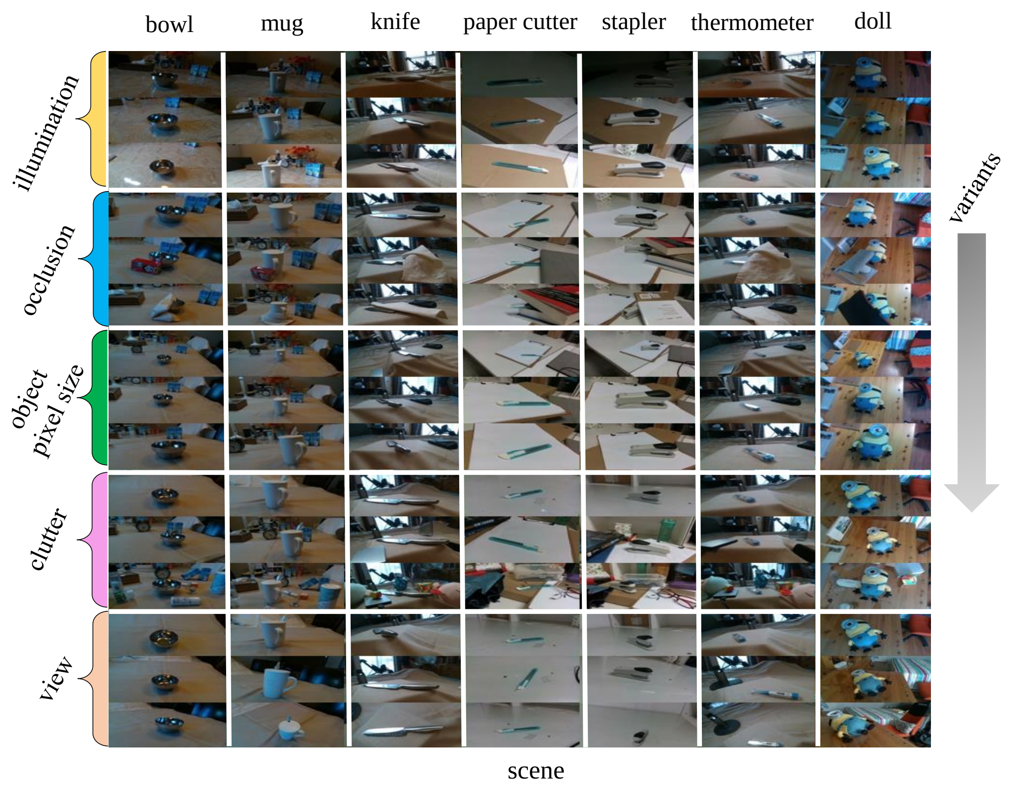

FIGURE 5.18: Picked samples of the objects from 7 scenes (column) under multiple level environment conditions (row). The variants from top to bottom are illumination (weak, normal, and strong); occlusion (0%, 25%, and 50%); object pixel size ($< 30 \times 30$, $30 \times 30 - 200 \times 200$, and $> 200 \times 200$); clutter (simple, normal and complex); and multi-views of the objects. (Note that we use different views as training samples of each difficulty level in each factor).

source codes or binary codes to evaluate their whole lifelong learning process to make fair comparison. The finalists' methods were tested by the organizers on Intel Core i9 CPU and 1Nvidia RTX 1080 Ti GPU. For final round dataset, we randomly shuffled the dataset with multiple factors. Data is split up to 12 batches/tasks and each batch/task samples are from one subdirectories (there are 12 subdirectories in total, 4 factors $\times$ 3 level/factor). Each batch includes 69 instances from 7 scenes, about 21520 test samples, 21,520 validation samples and 172,200 training samples. The metrics and corresponding grading weights are shown in Table 5.7. As can be seen, we also provided a bonus test set which is recorded in under different context background with some deformation. The adaptation on this bonus testing data is a challenging task for our task.

Metric	Accuracy	Model Size	Inference Time	Replay Size
Weight	50%	8%	8%	8%

Metric	Oral Presentation	Accuracy on Bonus Dataset
Weight	10%	16%

TABLE 5.7: Metrics and grading criteria for final round.

Challenge Results From more than 150 registered participants, 8 teams entered in the final phase and submitted results, codes, posters, slides and abstract papers (available here). Table 5.8 reports the details of all metrics (except oral presentation) for each team.

Architectures and main ideas: All the proposed methods use end-to-end deep learning models and employ the GPU(s) for training. For lifelong learning strategies: 5 teams applied regularization methods, 2 teams utilized knowledge distillation methods and 1 team used network expansion method. 4 teams applied resampling mechanism to alleviate catastrophic forgetting. Meanwhile, some other computer vision methods including saliency map, Single Shot multi-box Detection (SSD), data augmentation are also utilized in their solutions.

Teams	Final Acc. (%)	Model Size (MB)	Inference time (s)	Replay Size (#sample)	Bonus-set Acc. (%)
HIK_ILG	96.86	16.30	25.42	**0**	**21.86**
Unibo	97.68	**5.900**	**22.41**	1,500	8.500
Guiness	72.90	9.400	346.0	**0**	10.96
Neverforget	92.93	342.9	467.1	**0**	1.520
SDU_BFA_PKU	**99.56**	171.4	2,444	28,500	19.54
Vidit98	96.16	9.400	112.2	1,300	1.390
HYDRA-DI-ETRI	10.42	13.40	1,323	21,312	7.100
NTU_LL	93.56	467.1	4,213	**0**	2.100

TABLE 5.8: IROS 2019 Lifelong Robotic Vision Challenge final results.

Challenge Methods and Teams

HIK_ILG Team The team developed the dynamic neural network, which was comprised of two parts: dynamic network expansion for data across dissimilar domains and knowledge distillation for data in similar domains (See Figure 5.19). They froze the shared convolutional layers and trained new heads for new tasks. The domain gap was determined by measuring the accuracy of the previous model before training on current task. In order to increase the generalization ability of the trained model, they used ImageNet pre-trained model for the shared convolutional layers, and took more data augmentation and more batches to train head1 for base model. Without using previous data, they discovered known instances in current task by a single forward pass via previous model. Those correctly classified were treated as known samples. They used these samples for knowledge distillation. They utilized the best head over multiple heads for distillation, which is verified by experimental results.

Unibo Team The team proposed a new Continual Learning approach based on latent rehearsal, namely the replay of latent neural network activation instead of raw images at the input level. The algorithm can be deployed on the edge with low latency. With latent rehearsal they denoted an approach where instead of maintaining in the external memory copies of input patterns in the form of raw data, they stored the pattern activation at a given level (denoted as latent rehearsal layer). The algorithm can be summarized as follow: 1) Take n patterns from the current batch; 2) Forward them through the network until the rehearsal layer; 3) Select k patterns from the rehearsal memory; 4) Concat the original and the replay patterns; 5) Forward all the patterns through the rest of the network; 6) Backpropagate the loss only until the rehearsal layer.

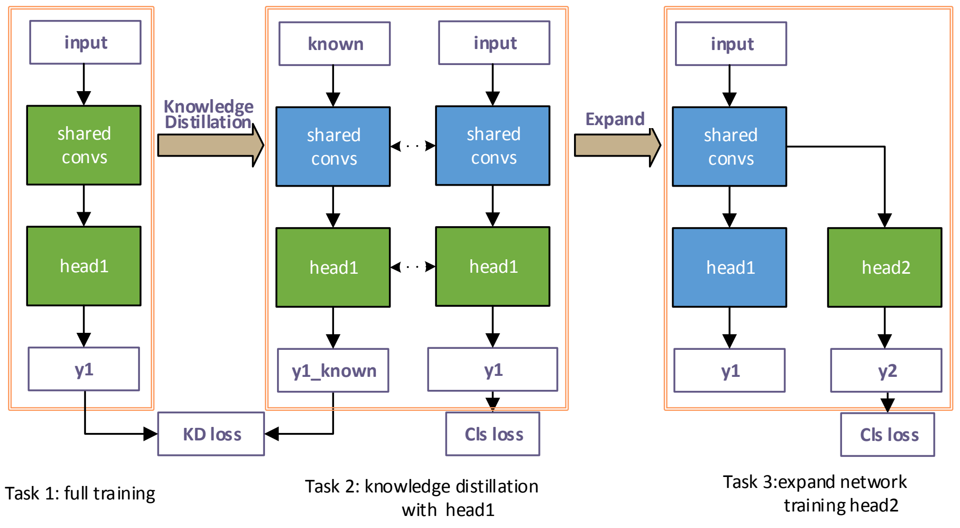

FIGURE 5.19: The architecture of proposed dynamic neural network by HIK_ILG Team.

The specific design they utilized with was AR1*, AR1*free and LwF CL approaches over a MobileNet-v1 and MobileNet-v2 [149, 113, 103, 66]. Meanwhile, they opted for simplicity and the trivial rehearsal approach summarized in Algorithm 4 is used for memory management.

Algorithm 4 Pseudo-code explaining how the external memory M is populated across the training batches.

```
Require: $M = \emptyset$
Require: $M_{size}$ = number of patterns to be stored in $M$
For each training batch $B_i$ do
    train the model on shuffled $B_i \cup M$
    $h = M_{size}/i$
    $R_{add}$ = Random sampling $h$patterns from $B_i$
    $R_{replace} = \begin{cases} \text{Sample } h \text{ patterns from } M, & \text{if } i > 1 \\ \emptyset, & \text{Otherwise} \end{cases}$
    $M = (M - R_{replace}) \cup R_{add}$
end for
```

The full version of this proposed lifelong learning method can be found here with an Android App demo for continual object recognition at the edge demo on this YouTube link [121].

5.4.2 The 1st CLVision Challenge at CVPR 2020

The first Continual Learning in Computer Vision challenge held at CVPR in 2020 has been one of the first opportunities to evaluate different continual learning algorithms on a common hardware with a large set of shared evaluation metrics and 3 different settings based on the realistic CORe50 video benchmark. In this section, we report the main results of the competition, which counted more than 79 teams registered, 11 finalists and 2300$ in prizes. We also summarize the winning approaches, current challenges and future research directions.

Background

Gradient-based architectures, such as neural networks trained with Stochastic Gradient Descent (SGD), notably suffer from catastrophic forgetting or interference [107, 142, 45], where the network parameters are rapidly overwritten when learning over non-stationary data distributions to model only the most recent. In the last few years, significant progresses have been made to tame the issue. Nevertheless, comparing continual learning algorithms today constitutes a hard task [35]. This is mainly due to the proliferation of different settings only covering partial aspects of the continual learning paradigm, with diverse training and evaluation protocols, metrics and datasets used [87, 18]. Another important question is whether such algorithms, that have mostly been proved on artificial benchmarks such as MNIST [86] or CIFAR [81], can scale and generalize to different settings and real-world applications.

The *1st Continual Learning in Computer Vision Challenge*, organized within the *CLVision* workshop at CVPR 2020, is one of the first attempts to address these questions. In particular, the main objectives of the competition were:

- Invite the research community to scale up continual learning approaches to natural images and possibly on video benchmarks.
- Invite the community to work on solutions that can generalize over multiple continual learning protocols and settings (e.g. with or without a "task" supervised signal).
- Provide the first opportunity for a comprehensive evaluation on a shared hardware platform for a fair comparison.

Notable competitions previously organized in this area include: the Pascal 2 EU network of excellence challenge on *"covariate shift"*, organized in 2005 [130, 129]; the *Autonomous Lifelong Machine Learning with Drift* challenge organized at NeurIPS 2018 [38] and the *IROS 2019 Lifelong Robotic Vision* challenge [9]. While the first two competitions can be considered as the first continual learning challenges ever organized, they were based on low-dimensional features benchmarks that made it difficult to understand the scalability of the proposed methods to more complex settings with deep learning based techniques. The latest competition, instead, has been one of the first challenges organized within robotic vision realistic settings. However, it lacked a general focus on computer vision applications as well as a comprehensive evaluation on 3 different settings and 4 tracks.

For transparency and reproducibility, we openly release the finalists' dockerized solutions as well as the initial baselines at the following link: `https://github.com/vlomonaco/cvpr_clvision_challenge`.

Competition

The *CLVision competition* was planned as a 2-phase event (pre-selection and finals), with 4 tracks and held online from the *15th of February 2020* to the *14th of June 2020*. The *pre-selection* phase, based on the codalab online evaluation framework[11], lasted 78 days and was followed by the finals where a dockerized solution had to be submitted for remote evaluation on a shared hardware. In the following section, the dataset, the different tracks, the evaluation metric used and the main rules of the

[11] `https://codalab.org`

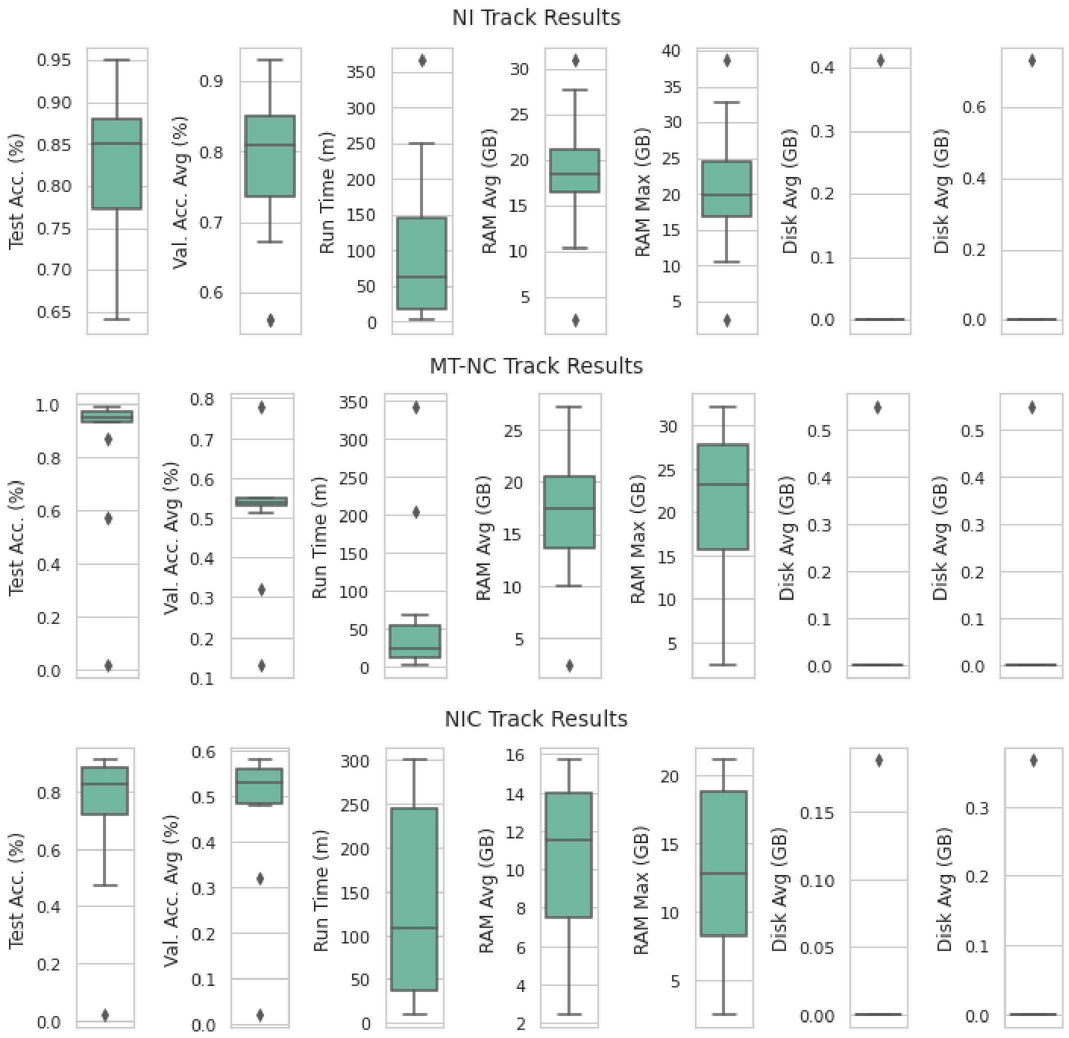

FIGURE 5.20: Results distributions for the three tracks (NI, MT-NC and NIC) across the 11 finalists solutions and the main evaluation metrics used for the competition: total test accuracy (%) at the end of the training, average validation accuracy over time (%), maximum and average RAM/Disk usage (GB).

competition are reported in detail. Finally, the main competition statistics, participants and winners are presented.

Dataset The CORe50 dataset described in [95] is used. An-in depth discussion on this dataset and related benchmarks can be found in Chapter 2. Classification on CORe50 can be performed at the instance (object) level (50 classes) or category level (10 classes). The former, being a more challenging task, was the configuration chosen for this competition. The egocentric vision of hand-held objects allows for the emulation of a scenario where a robot has to incrementally learn to recognize objects while manipulating them. Objects are presented to the robot by a human operator who can also provide the labels, thus enabling a supervised classification (such an applicative scenario is well described in [117, 158]).

Tracks Based on the CORe50 dataset, the challenge included four different tracks based on the different settings considered:

Team Name	Team Members
HaoranZhu	Haoran Zhu
ICT_VIPL	Chen He, Qiyang Wan, Fengyuan Yang, Ruiping Wang, Shiguang Shan, Xilin Chen
JimiB	Giacomo Bonato, Francesco Lakj, Alex Torcinovich, Alessandro Casella
Jodelet	Quentin Jodelet, Vincent Gripon, Tsuyoshi Murata
Jun2Tong	Junyong Tong, Amir Nazemi, Mohammad Javad Shafiee, Paul Fieguth
MrGranddy	Vahit Bugra Yesilkaynak, Firat Oncel, Furkan Ozcelik, Yusuf Huseyin Sahin, Gozde Unal
Noobmaster	Zhaoyang Wu, Yilin Shao, Jiaxuan Zhao, and Bingnan Hu
Sahinyu	Yusuf H. Sahin, Furkan Ozcelik, Firat Oncel, Vahit Bugra Yesilkaynak, Gozde Unal
Soony	Soonyong Song, Heechul Bae, Hyonyoung Han, Youngsung Son
UT_LG	Zheda Mai, Hyunwoo Kim, Jihwan Jeong, Scott Sanner
YC14600	Yu Chen, Jian Ma, Hanyuan Wang, Yuhang Ming, Jordan Massiah, Tom Diethe

TABLE 5.9: The 11 finalists of the 1st CLVision Competition.

1. New Instances (NI): In this setting 8 training batches of the same 50 classes are encountered over time. Each training batch is composed of different images collected in different environmental conditions.
2. Multi-Task New Classes (MT-NC)[12]: In this setting the 50 different classes are split into 9 different tasks: 10 classes in the first batch and 5 classes in the other 8. In this case the task label will be provided during training and test.
3. New Instances and Classes (NIC): this protocol is composed of 391 training batches containing 300 images of a single class. No task label will be provided and each batch may contain images of a class seen before as well as a completely new class.
4. All together (ALL): All the settings presented above.

Each participant of the challenge could choose in which of the main three tracks (NI, MT-NC, NIC) to compete. Those participants that decided to participate to all the three main tracks were automatically included in the ALL track as well, the most difficult and ambitious track of the competition.

Evaluation Metric In the last few years, the main evaluation focus in continual learning has always been centered around accuracy-related forgetting metrics. However, as argued by [35], this may lead to biased conclusion not accounting for the real scalability of such techniques over an increasing number of tasks/batches and more complex settings. For this reason, in the competition each solution was evaluated across a number of metrics:

[12]Multi-Task-NC constitutes a simplified variation of the originally proposed New Classes (NC) protocol [95] (where the task label is not provided during train and test).

1. *Final accuracy on the test set*[13]: computed only at the end of the training procedure.
2. *Average accuracy over time on the validation set*: computed at every batch/task.
3. *Total training/test time*: total running time from start to end of the main function (in minutes).
4. *RAM usage*: total memory occupation of the process and its eventual subprocesses. It is computed at every epoch (in MB).
5. *Disk usage*: only of additional data produced during training (like replay patterns) and additionally stored parameters. It is computed at every epoch (in MB).

The final aggregation metric (CL_{score}) is the weighted average of the 1-5 metrics (0.3, 0.1, 0.15, 0.125, 0.125 respectively).

Rules and Evaluation Infrastructure In order to provide a fair evaluation while not constraining each participants to simplistic solutions due to a limited server-side computational budget, the challenge was based on the following rules:

1. The challenge was based on the *Codalab* platform. For the pre-selection phase, each team was asked to run the experiments *locally* on their machines with the help of a Python repository to easily load the data and generate the submission file (with all the necessary data to execute the submission remotely and verify the adherence to the competition rules if needed). The submission file, once uploaded, was used to compute the CL_{Score} which determined the ranking in each scoreboard (one for each track).
2. It was possible to optimize the data loader, but not to change the data order or the protocol itself.
3. The top 11 teams in the scoreboard at the end of the pre-selection phase were selected for the final evaluation.
4. The final evaluation consisted in a *remote* evaluation of the final submission for each team. This is to make sure the final ranking was computed in the same computational environment for a fair comparison. In this phase, experiments were run remotely for all the teams over a 32 CPU cores, 1 NVIDIA Titan X GPU, 64 GB RAM Linux system. The max running time was capped at 5 hours for each submission/track.
5. Each team selected for the final evaluation had to submit a single dockerized solution which had to contain the exact same solution submitted for the last codalab evaluation. The initial docker image (provided in the initial challenge repository) could have been customized at will but without exceeding 5 GB.

It is worth noting that only the test accuracy was considered in the ranking of the pre-selection phase of the challenge, since the evaluation was run on participants' local hardware. However, since it was not possible to submit a different solution for the final evaluation, this ensured the competition was not biased on the sole accuracy metric.

[13]Accuracy in CORe50 is computed on a fixed test set. Rationale behind this choice is explained in [95]

The financial budget for the challenge was entirely allocated for the monetary prizes in order to stimulate participation:

- 800$ for the participant with highest average score accross the three tracks (e.g the ALL track).
- 500$ for the participant with highest score on the NI track.
- 500$ for the participant with highest score on the MT-NC track.
- 500$ for the participant with highest score on the NIC track.

These prizes were kindly sponsored by *Intel Labs (China)*, while the remote evaluation was performed thanks to the hardware provided by the *University of Bologna*.

Participants and Finalists The challenge counted the participation of 79 teams worldwide that competed during the pre-selection phase. From those 79 teams only 11 qualified to the finals with a total of 46 people involved and an average team components number of 4. In Table 5.9 the 11 finalist teams and their members are reported.

Continual Learning Approaches

In this section we discuss the baselines made available as well as the continual learning approaches of the winning teams in more details. On the official competition website an extended report for each of the finalist team detailing their approach is also publicly available.[14]

Baselines In order to better understand the challenge complexity and the competitiveness of the proposed solutions, three main baselines were included for each of the 4 tracks:

- *Naive*: This is the basic *finetuning* strategy, where the standard SGD optimization process is continued on the new batches/tasks without any additional regularization constraint, architectural adjustment or memory replay process.
- *Rehearsal*: In this baseline the *Naive* approach is augmented with a basic replay process with a growing external memory, where 20 images for each batch are stored.
- *AR1* with Latent Replay*: a recently proposed strategy [121] showing competitive results on CORe50 with a shared, non fine-tuned hyper-parametrization across the three main tracks.

Team ICT_VIPL **General techniques for all tracks**. To improve their performance the ICT_VIPL team used: (1) *Heavy Augmentation* with the Python `imgaug` library[15]; (2) resize the input image to 224$\times$224 to encourage more knowledge transfer from the ImageNet pretrained model; (3) employ an additional exemplar memory for episodic memory replay to alleviate catastrophic forgetting (randomly select 2 $\sim$ 3% of the training samples); (4) striking a balance between performance and model capacity by using a moderately deep network ResNet-50. As for efficiency, they leveraged the PyTorch Dataloader module for multi-thread speed-up.

[14] `https://sites.google.com/view/clvision2020/challenge`
[15] `https://imgaug.readthedocs.io`

Special techniques for individual tracks. For NI track, there is no special design over the general techniques above and they only tune the best hyper-parameters. For Multi-Task-NC track, they carefully design a pipeline that disentangles representation and classifier learning, which shows very high accuracy and the pipeline is as below (D_i is the set of exemplars for Task i and $|D_i|$ is its size):

For Task 0: (1) Train the feature extractor $f(x)$ and the first head $c_0(z)$ with all training samples; (2) Select N samples randomly and store them in the exemplar memory ($|D_0| = N$).

For Task i ($i = 1, 2, \dots, 8$): (1) Train head $c_i(z)$ with all training samples of Task i; (2) Drop some samples randomly from the previous memory, keep $|D_j| = \frac{N}{i+1}$ (for all $j < i$); (3) Select $\frac{N}{i+1}$ samples from Task i randomly and store them in the exemplar memory ($|D_i| = \frac{N}{i+1}$); (4) Fine-tune the feature extractor $f(x)$ with all samples in the memory $\cup_j D_j (j \leq i)$. (since the feature extractor alone cannot classify images, a temporary head $c(z)$ is used for training); (5) Fine-tune each head $c_j(z)$ with the corresponding samples in the memory $D_j (j \leq i)$.

For NIC track, based on the assumption that the neural network estimates Bayesian a posteriori probilitities [139], the network outputs are divided by the prior probability for each class inspired by the trick that handles class imbalance [15]. Such a technique can prevent the classifier from biasing minority class (predict to newly added classes) especially in the first few increments.

Team Jodelet The proposed solution consists in the concatenation of a pre-trained deep convolutional neural network used as a feature extractor and an online trained logistic regression combined with a small reservoir memory [21] used for rehearsal.

Since the guiding principle of the proposed solution is to limit as much as possible the computational complexity, the model is trained in an online continual learning setting: each training example is only used once. In order to further decrease the memory and computational complexity of the solution at the cost of a slight decrease of the accuracy, the pre-trained feature extractor is fixed and is not fine-tuned during the training procedure. As a result, it is not necessary to apply the gradient descent algorithm to the large feature extractor and the produced representation is fixed. Therefore, it is possible to store the feature representation in the reservoir memory instead of the whole input raw image. In addition to the memory gain, this implies that the replay patterns do not have to go through the feature extractor again, effectively decreasing the computational complexity of the proposed solution.

Among the different architectures and training procedures considered for the feature extractor, ResNet-50 [61] trained by Facebook AI using the Semi-Weakly Supervised Learning procedure [186] was selected. This training procedure relies on the use of a teacher model and 940 million public images in addition to the ImageNet dataset [145]. Compared with the reference training procedure in which the feature extractor is solely trained on the ImageNet dataset, this novel training procedure allows for a consequent increase of the accuracy without modifying the architecture: while the difference of Top-1 accuracy between both training procedures for ResNet-50 is about 5.0% on Imagenet, the difference increases up to 11.1% on the NIC track of the challenge. Moreover, it should be noted that on the three tracks of the challenge, ResNet-18 feature extractor trained using this new procedure is able to reach an accuracy comparable with the one of the reference ResNet-50 feature extractor trained only on ImageNet, while being considerably smaller and faster.

For reasons of consistency, the same hyperparameters have been used for the three tracks of the challenge and have been selected using a grid search.

Team UT_LG Batch-level Experience Replay with Review In most *Experience Replay* based methods, the incoming mini-batch is concatenated with another mini-batch of samples retrieved from the memory buffer. Then, they simply takes an SGD step with the concatenated samples, followed by an update of the memory [21, 17]. Team UT_LG method makes two modifications. Firstly, to reduce the number of retrieval and update steps, they concatenate the memory examples at the batch level instead of at the mini-batch level. Concretely, for every epoch, they draw a batch of data $D_{\mathcal{M}}$ randomly from memory with size *replay_sz*, concatenate it with the current batch and conduct the gradient descent parameters update. Moreover, they add a review step before the final testing, where they draw a batch of size D_R from memory and conduct the gradient update again. To prevent overfitting, the learning rate in the review step is usually lower than the learning rate used when processing incoming batches. The overall training procedure is presented in Algorithm 2. **Data Preprocessing** (1) Centering-cropping the image with a (100, 100) window to make the target object occupy more pixels in the image. (2) Resizing the cropped image to (224, 224) to ensure no size discrepancy between the input of the pre-trained model and the training images. (3) Pixel-level and spatial-level data augmentation to improve generalization. The details of their implementation can be found in [100]

```
Procedure BERR$\mathcal{D}$, mem_sz, replay_sz, review_sz, lr_replay, lr_review
    $\mathcal{M} \leftarrow \{\} * mem\_sz$
    for $t \in \{1, \ldots, T\}$ do
        for epochs do
            if $t > 1$ then
                $D_{\mathcal{M}} \overset{\text{replay\_sz}}{\sim} \mathcal{M}$
                $D_{\text{train}} = D_{\mathcal{M}} \cup D_t$
            else
                $D_{\text{train}} = D_t$
            end
            $\theta \leftarrow \text{SGD}(D_{\text{train}}, \theta, \text{lr\_replay})$
        end
        $\mathcal{M} \leftarrow UpdateMemory(D_t, \mathcal{M}, \text{mem\_sz})$
    end
    $D_R \overset{\text{review\_sz}}{\sim} \mathcal{M}$
    $\theta \leftarrow \text{SGD}(D_R, \theta, \text{lr\_review})$
    return $\theta$
```

Algorithm 2: Batch-level Experience Replay with Review

Team Yc14600 The use of episodic memories in continual learning is an efficient way to prevent the phenomenon of *catastrophic forgetting*. In recent studies, several gradient-based approaches have been developed to make more efficient use of compact episodic memories. The essential idea is to use gradients produced by samples from episodic memories to constrain the gradients produced by new samples, *e.g.* by ensuring the inner product of the pair of gradients is non-negative [99] as follows:

$$\langle g_t, g_k \rangle = \left\langle \frac{\partial \mathcal{L}(x_t, \theta)}{\partial \theta}, \frac{\partial \mathcal{L}(x_k, \theta)}{\partial \theta} \right\rangle \geq 0, \forall k < t \tag{5.1}$$

where t and k are time indices, x_t denotes a new sample from the current task, and x_k denotes a sample from the episodic memory. Thus, the updates of parameters are forced to preserve the performance on previous tasks as much as possible. 5.1 indicates larger cosine similarities between gradients produced by current and previous tasks result in improved generalisation. This in turn indicates that samples that lead to the most diverse gradients provide the most difficulty during learning.

Through empirical studies the team members found that the discrimination ability of representations strongly correlates with the diversity of gradients, and more discriminative representations lead to more consistent gradients. They use this insight to introduce an extra objective Discriminative Representation Loss (DRL) into the optimization objective of classification tasks in continual learning. Instead of explicitly refining gradients during training process, DRL helps with decreasing gradient diversity by optimizing the representations. As defined in 5.2, DRL consists of two parts: one is for minimizing the similarities of representations between samples from different classes ($\mathcal{L}_{bt}$), the other is for minimizing the similarities of representations between samples from a same class ($\mathcal{L}_{wi}$) for preserving information of representations for future tasks.

$$\begin{aligned} &\min_{\Theta} \mathcal{L}_{DR} = \min_{\Theta} (\mathcal{L}_{bt} + \mathcal{L}_{wi}), \\ &\mathcal{L}_{bt} = \frac{1}{B_{bt}} \sum_{l=1}^{L} \sum_{i=1}^{B} \sum_{j=1, y_j \neq y_i}^{B} \langle h_{l,i}, h_{l,j} \rangle, \\ &\mathcal{L}_{wi} = \frac{1}{B_{wi}} \sum_{l=1}^{L} \sum_{i=1}^{B} \sum_{j=1, j \neq i, y_j = y_i}^{B} \langle h_{l,i}, h_{l,j} \rangle. \end{aligned} \tag{5.2}$$

where Θ denotes the parameters of the model, L is the number of layers of the model, B is training batch size. B_{bt} and B_{wi} denote the number of pairs of samples in the training batch that are from different classes and the same class, respectively, $h_{l,i}$ is the output of layer l by input x_i and y_i is the label of x_i. Please refer to [24] for more details.

Competition Results

In this section we detail the main results of the competition for each of the main three tracks (NI, MT-NC & NIC) as well as the averaged track ALL, which determined the overall winner of the challenge. For each track the teams are ranked as follows: i) each metric is normalized across between 0 and 1; ii) the CL_{score} is computed as a weighted average; ii) results are ordered in descending order.

In the next sections we report the results with their absolute values to better grasp the quality of the solutions proposed and their portability in different applicative contexts.

New Instances (NI) Track In Table 5.10 the main results for the *New Instances* (NI) track are reported. In Table C.1, additional details (not taken into account for the evaluation) for each solution are shown. In this track, the *UT_LG* obtained the best

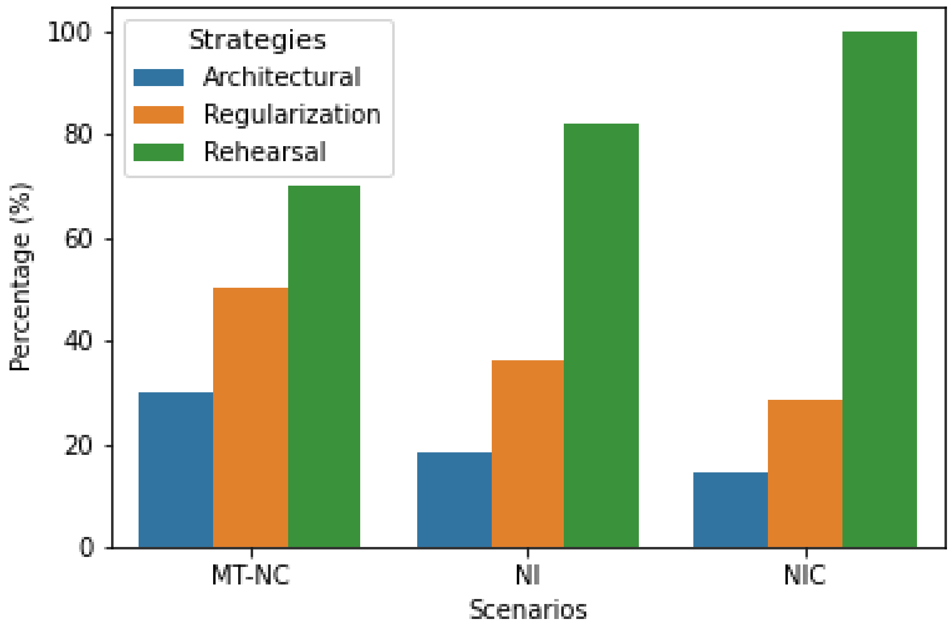

FIGURE 5.21: Percentage (%) of finalists solutions for each track employing an *architectural*, *regularization* or *rehearsal* strategy. Percentages do not sum to 100% since many approached used hybrid strategies. Better viewed in colors.

CL_{Score} with a small gap w.r.t. its competitors. The test accuracy tops 91% for the winning team, showing competitive performance also in real-world non-stationary applications. It is worth noting that the top-4 solutions all employed a rehearsal-based technique, only in one case supported by a regularization counterpart.

Multi-Task NC (MT-NC) Track For the MT-NC track, results are reported in Table 5.11 and additional details in Table C.2 of the Appendix. In this scenario, arguably the easiest since it provided an additional supervised signal (the Task label) the AR1 baseline resulted as the best scoring solution. In fact, while achieving lower accuracy results than the other top-7 solutions, it offered a more efficient algorithmic proposal in terms of both memory and computation (even without a careful hyper-parametrization). It is also interesting to note that, in this scenario, it is possible to achieve impressive accuracy performance (∼99%) within reasonable computation and memory constraints as shown by the ICT_VIPL team, the only solution who opted for a disk-based exemplars memorization.

New Instances (NIC) Track The NIC track results are reported in Table 5.12. Additional details of each solution are also made available in Table C.3. Only 7 over 11 finalist teams submitted a solution for this track. In this case, it is possible to observe generally lower accuracy results and an increase in the running times across the 391 batches.

All (ALL) Track Finally in Table 5.13 the results averaged across tracks are reported for the ALL scoreboard. Also in this case the competing teams were 7 over a total of 11 with *UT_LG* as the winning team. With an average testing accuracy of ∼92%, a average memory consumption of ∼10 GB and a running time of ∼68 minutes, its

TEAM NAME	TEST ACC (%)	VAL ACC_{avg} (%)	RUN_{time} (M)	RAM_{avg} (MB)	RAM_{max} (MB)	$DISK_{avg}$ (MB)	$DISK_{max}$ (MB)	CL_{score}
UT_LG	0.91	0.90	63.78	11429	11643	0	0	0.692
YC14600	0.88	0.85	22.58	17336	18446	0	0	0.648
ICT_VIPL	0.95	0.93	113.70	2459	2460	422	750	0.629
JODELET	0.84	0.85	3.11	18805	18829	0	0	0.612
SOONY	0.85	0.81	25.57	16662	17000	0	0	0.602
JIMIB	0.91	0.89	248.82	19110	25767	0	0	0.573
JUN2TONG	0.84	0.76	62.48	20968	23252	0	0	0.550
SAHINYU	0.88	0.81	156.64	26229	32176	0	0	0.538
AR1	0.75	0.73	17.18	10550	10838	0	0	0.520
NOOBMASTER	0.85	0.75	74.54	31750	39627	0	0	0.504
MRGRANDDY	0.88	0.84	249.28	28384	33636	0	0	0.501
NAIVE	0.66	0.56	2.61	18809	18830	0	0	0.349
REHEARSAL	0.64	0.56	3.79	21685	21704	0	0	0.326
HAORANZHU	0.70	0.67	366.22	21646	21688	0	0	0.263
AVG	0.82	0.78	100.74	18987	21135	30.13	53.57	0.52

TABLE 5.10: NI track results for the 11 finalists of the competition and the three baselines.

relatively simple solution suggests continual learning for practical object recognition applications to be feasible in the real-world, even with a large number of small non-i.i.d. bathes.

Discussion Given the main competition results and the additional solutions details reported in Appendix C, we can formulate a number of observations to better understand current issues, consolidated approaches and possible future directions for competitive continual learning algorithms tested on real-world computer vision applications.

In particular, we note:

- *Different difficulty for different scenarios*: averaging the 11 finalists test accuracy results we can easily deduce that the MT-NC track or scenario was easier than the NI one (∼85% vs ∼82%), while the NIC track was the most difficult with a average accuracy of ∼72%. This is not totally surprising, considering that the MT-NC setting allows access to the additional task labels and the NI scenario does not include dramatic distributional shifts, while the NIC one includes a substantially larger number of smaller training batches. Moreover, a number of researchers already pointed out how different training/testing regimes impacts forgetting and the continual learning process [110, 103, 57].
- *100% of the teams used a pre-trained model*: All the solutions, for all the tracks started from a pre-trained model on ImageNet. While starting from a pre-trained model is notably becoming a standard for real-world computer vision applications, we find it interesting to point out such a pervasive use in the challenge. While this does not mean pre-trained model should be used for every continual learning algorithm in general, it strongly suggests that for solving real-world computer vision application today, pre-training is mostly needed.
- *∼90% of the teams used a rehearsal strategy*: rehearsal constitutes today one of the easiest and effective solution to continual learning where previous works

TEAM NAME	TEST ACC (%)	VAL ACC$_{avg}$ (%)	RUN$_{time}$ (M)	RAM$_{avg}$ (MB)	RAM$_{max}$ (MB)	DISK$_{avg}$ (MB)	DISK$_{max}$ (MB)	CL_{score}
AR1	0.93	0.53	16.02	10263	14971	0	0	0.693
UT_LG	0.95	0.55	19.02	13793	16095	0	0	0.691
YC14600	0.97	0.54	11.81	15870	19403	0	0	0.686
SOONY	0.97	0.55	55.02	14005	16049	0	0	0.679
JODELET	0.97	0.55	2.55	17893	23728	0	0	0.679
JUN2TONG	0.96	0.55	28.80	18488	19588	0	0	0.671
ICT_VIPL	0.99	0.55	25.20	2432	2432	562	562	0.630
REHEARSAL	0.87	0.51	4.49	20446	28329	0	0	0.626
JIMIB	0.95	0.78	204.56	21002	24528	0	0	0.607
MRGRANDDY	0.94	0.54	46.52	27904	32921	0	0	0.604
NOOBMASTER	0.95	0.53	68.07	27899	32910	0	0	0.597
HAORANZHU	0.57	0.32	343.50	21223	28366	0	0	0.351
NAIVE	0.02	0.13	3.41	17897	23726	0	0	0.318
AVG	0.85	0.51	63.77	17624	21773	43.27	43.27	0.60

TABLE 5.11: NC track results for the 11 finalists of the competition and the three baselines. Teams not appearing in the table did not compete in this track.

TEAM NAME	TEST ACC (%)	VAL ACC$_{avg}$ (%)	RUN$_{time}$ (M)	RAM$_{avg}$ (MB)	RAM$_{max}$ (MB)	DISK$_{avg}$ (MB)	DISK$_{max}$ (MB)	CL_{score}
UT_LG	0.91	0.58	123.22	6706	7135	0	0	0.706
JODELET	0.83	0.54	14.12	10576	11949	0	0	0.694
AR1	0.71	0.48	28.19	3307	4467	0	0	0.693
ICT_VIPL	0.90	0.56	91.29	2485	2486	192	375	0.625
YC14600	0.89	0.57	160.24	16069	21550	0	0	0.586
REHEARSAL	0.74	0.50	60.32	15038	19488	0	0	0.585
SOONY	0.82	0.52	280.39	12933	14241	0	0	0.533
JIMIB	0.87	0.56	272.98	13873	21000	0	0	0.533
NOOBMASTER	0.47	0.32	300.15	14492	18262	0	0	0.346
NAIVE	0.02	0.02	9.45	10583	11917	0	0	0.331
AVG	0.72	0.47	134.03	10606	13249	19.22	37.50	0.56

TABLE 5.12: NIC track results for the 11 finalists of the competition and the three baselines. Teams not appearing in the table did not compete in this track.

[58] have shown that even a very small percentage of previously encountered training data can have huge impacts on the final accuracy performance. Hence, it is not surprising that a large number of teams opted to use it for maximizing the CL_{score}, which only slightly penalized its usage.

- *~45% of the teams used a regularization approach*: regularization strategies have been extensively used in the competition. It worth noting though, that only 1 team used it alone and not in conjunction with a plain rehearsal or architectural approaches.
- *only ~27% of the teams used an architectural approach*: less then one third of the participants did use an architectural approach but only on conjunction with a rehearsal or regularization one. This evidence reinforces the hypothesis that architectural-only approaches are difficult to scale efficiently over a large number of tasks or batches [147].

TEAM NAME	TEST ACC (%)	VAL ACC$_{avg}$ (%)	RUN$_{time}$ (M)	RAM$_{avg}$ (MB)	RAM$_{max}$ (MB)	DISK$_{avg}$ (MB)	DISK$_{max}$ (MB)	CL_{score}
UT_LG	0.92	0.68	68.67	10643	11624	0	0	0.694
JODELET	0.88	0.64	6.59	15758	18169	0	0	0.680
AR1	0.80	0.58	20.46	8040	10092	0	0	0.663
YC14600	0.91	0.65	64.88	16425	19800	0	0	0.653
ICT_VIPL	0.95	0.68	76.73	2459	2459	392	562	0.617
SOONY	0.88	0.63	120.33	14533	15763	0	0	0.612
REHEARSAL	0.75	0.52	22.87	19056	23174	0	0	0.570
JIMIB	0.91	0.74	242.12	17995	23765	0	0	0.542
NOOBMASTER	0.76	0.53	147.59	24714	30266	0	0	0.464
NAIVE	0.23	0.24	5.16	15763	18158	0	0	0.327
AVG	0.80	0.59	77.54	14539	17327	39.22	56.25	0.58

TABLE 5.13: ALL track results for the 11 finalists of the competition and the three baselines. Teams not appearing in the table did not compete in this track.

- *Increasing replay usage with track complexity*: as shown in Figure 5.21, it is worth noting that as the track complexity increased, the proposed solutions tended to include more replay mechanisms. For example, for the NIC track, all the approaches included rehearsal, often used in conjunction with a regularization or architectural approach.
- *High memory replay size*: it is interesting to note that many CL solutions employing rehearsal have chosen to use a *growing* memory replay buffer rather than a fixed one with an average maximum memory size (across teams and tracks) of ∼26k patterns. This is a very large number considering that is about ∼21% of the total CORe50 training set images.
- *Different hyper-parameters selection*: An important note to make is about the hyperparameters selection and its implication to algorithms *generalization* and *robustness*. Almost all participants' solutions involved a carefully fine-tuned hyper-parameters selection which was different based on the continual scenario tackled. This somehow highlights the weakness of state-of-the-art algorithms and their inability to truly generalize to novel situations never encountered before. A notably exception is the *AR1* baseline, which performed reasonably well in all the tracks with a shared hyperparametrization.

Conclusions and Future Improvements

The *1st Continual Learning for Computer Vision Challenge* held at CVPR2020 has been one of the first large-scale continual learning competition ever organized with a raised benchmark complexity and targeting real-word applications in computer vision. This challenge allowed every continual learning algorithm to be fairly evaluated with shared and unifying criteria and pushing the CL community to work on more realistic benchmarks than the more common MNIST or CIFAR.

After a carefully investigation and analysis of the competition results we can conclude that continual learning algorithms are mostly ready to face real-world settings involving high-dimensional video streams. This is mostly thanks to hybrid approaches often combined with plain replay mechanisms. However, it remains unclear if such techniques can scale over longer data sequences and without such an extensive use of replay.

Despite the significant participation and success of the 1st edition of the challenge, a number of possible improvements and suggestions for future continual learning competitions can be formulated:

- *Discourage over-engineered solutions*: one of the main goal of the competition was to evaluate the applicability of current continual learning algorithms on real-world computer vision problems. However, given the substantial freedom given through the competition rules to achieve this goal, we have noticed a number of over-engineered solutions aimed at improving the CL_{score} but not really significant in terms of novelty of scientific interest. This in turns forced every other participants to focus on over-engineering rather than on the core continual learning issues. For example, data loading or compression algorithms may be useful to decrease memory and compute overheads but may be applicable to most of the solutions proposed, making them less interesting and out of the scope of competition. For this reason, we believe that finding a good trade-off between realism and scientific interest of the competition will be fundamental for future challenges in this area. We suggest for example to block the possibility to optimize the data loading algorithms and to count the number of replay patterns rather than their bytes overhead.
- *Automatize evaluation*: in the current settings of the challenge the evaluation was client-side (on the participants machines) for the pre-selection phase and on a server-side shared hardware for the finals. To ensure the fairness of the results and the competition rules adherence, the code that generated each submission had to be included as well. However, an always-available remote docker evaluation similar to the one proposed for the AnimalAI Olympics [28], would allow a single phase competition with an always coherent and updated scoreboard, stimulating in turns teams participation and retention over the competition period. This would also alleviate some burdens at the organization levels, reducing the amount of manual interventions.
- *Add scalability metrics*: An interesting idea to tame the challenge complexity while still providing a good venue for assessing continual learning algorithms advancement, would be to include other than the already proposed metrics, a number of derivative ones taking into account their trend over time rather than their absolute value. This would help to better understand their scalability on more complex problems and longer tasks/batches sequences and incentivize efficient solutions with constant memory/computation overheads.
- *Encourage the focus on original learning strategies*: Another important possible improvement of the competition would be setting up a number of incentives and disincentives to explore interesting research directions in continual learning. For example, the usage of pre-trained models has been extensively used for the competition by all the participants. However it would have been also interesting to see proposals not taking advantage of it as well. In the next competition edition we plan to discourage the use of pre-trained models, different hyperparameters for each setting track and increase the memory usage weight associated to the CL_{score}.

Chapter 6

Conclusions and Future Challenges

This chapter summarizes the contributions presented in this thesis. For each contribution, in the context of enabling continual learning capabilities in real-life applications, possible future research directions are discussed.

6.1 Realistic benchmarks

The scenarios discussed in Chapter 2 were designed with the idea of better modeling real-life situations. In particular, benchmarks of the NIC family introduce the idea that no assumptions on the possibility and frequency of encountering new and old concepts over time (*repetition of concepts*) should be made when designing continual learning strategies able to operate in realistic scenarios for embodied (and other environment-aware) agents. In this context, NICv2 was proposed, in which new classes of objects are inserted in random moments during the experiences stream. The NICv2 generation procedure was then applied to the CORe50 dataset [95] to produce the NICv2-79, -196, and -391 benchmarks. In particular, the latter one features the following elements of complexity, which made it an excellent reference throughout the thesis:

- a long stream of 391 incremental training experiences. As far as we know, no benchmark with such a high amount of experiences has been proposed in the literature;
- non-i.i.d. experiences. As experiences are composed of a single video depicting an object, the training instances (frames) are highly correlated. In addition, strategies have to account for a single-class composition of the training data, which poses an important challenge;
- instance-based recognition. We deem this element, which is also shared with NICv2-79 and -196 benchmarks, as a very important aspect of these benchmarks: while mainstream evaluation protocols focus on the category-based object recognition task, many practical applications may leverage instance recognition capabilities, in which specific objects are recognized in an environment.

Research directions The definition of realistic benchmarks is not an easy task. We argue that, while the NICv2 benchmarks here proposed move in the correct direction, much more can be done to close the gap between current artificial benchmarks and real scenarios. For instance, while both the original NIC [95] and the proposed NICv2 benchmarks successfully get rid of the *no-repetition* constraint, it may make sense to introduce elements able to better simulate realistic events, such as the sudden availability of abundant data about a certain class in a short time span, the fact that data about a class may be available again at a variable distance in time, the introduction of signals able to communicate the fact that remembering a certain concept is not required anymore, etcetera.

6.2 Effective continual learning techniques

In Chapter 3, efficient techniques such as AR1* and latent replay have been described. These techniques can effectively handle complex benchmarks such as NICv2-391. In addition, thanks to their flexibility, they can be seamlessly used in class-incremental and domain-incremental scenarios. Not only these techniques can outperform popular strategies such as iCaRL [135] and DSLDA [59] on the accuracy side, but they are also able to handle the training phase efficiently, thus allowing for their deployment on embedded, mobile, and robotic systems.

In particular, it was shown in Section 3.1 that replay-free techniques can be applied to handle complex benchmarks of the NICv2 family. This is a very important step able to prove that, even in the extreme case of 391 incremental experiences, replay is not strictly necessary to obtain a stable accuracy increase in time. With the introduction of Latent Replay (described in Section 3.2) the gap with the upper bound represented by the cumulative (joint training) approach is further reduced. With latent replay, we showed that replay can be highly beneficial even when instances are replayed at intermediate layers of the model. Moreover, we showed that this approach offers many accuracy/performance tradeoffs, which is ideal for practical applications. Such scalability is hardly found in mainstream techniques.

Research directions As described in Section 1.3.1, continual learning approaches are very varied and a precise direction for future works is hard to define. However, by taking the idea of developing real-life scenarios and applications as the main goal, a few macro-directions can be pointed out with enough confidence.

One of the main challenges in deep learning research is to effectively enable the use of abundant unlabeled data. While this is an open research direction in the deep learning field, it is even more relevant for continual learning research. In fact, real-life applications for embodied agents would be able to leverage a massive amount of unlabeled data coming from sensors. We argue that continual learning strategies should feature mechanisms able to improve the internal knowledge model incrementally by using this massive amount of data. These mechanisms should be able to cooperate with more classic supervised continual learning approaches to form a single effective learning system.

Another promising research direction is to design *expansion techniques* able to both expand and contract the model capacity in scenarios such as NIC, where the *no-repetition* idea does not hold, and when task labels and boundaries are not available at test time. We argue that proper capacity expansion techniques will be essential to enable lifelong learning capabilities in intelligent agents, no matter if they are embodied or server-powered ones.

6.3 Continual learning libraries

Chapter 4 opens with a discussion on the desiderata and design principles for a continual learning library. The Avalanche library described in Section 4.2 was designed to adhere to these principles. Its modularity and comprehensiveness allows for both new and expert users to quickly adopt the library as the base for their experimental setup. The current modules offered by the library (benchmarks, training, models, metrics, and logging) try to cover all aspects of a CL research codebase in a simple and integrated way. Avalanche was developed in a community effort inside the ContinualAI organization and it is one of the most popular continual learning research libraries.

Research directions Continual learning is a constantly evolving research field. Novel research directions, along with their scenarios, benchmarks, and techniques are regularly proposed in the literature. These new efforts are based on assumptions and tools that will hardly be already covered by an existing library. In other words, a successful continual learning library needs to be constantly expanded by adding new tools and modules. Thanks to the open contribution approach, the library is constantly evolving to accommodate these new needs. However, contributions regarding the deeper structure of the library usually come from a core team of maintainers.

In this context, the most natural evolution for Avalanche is to expand its capabilities in terms of support for *object detection and segmentation*, *continual unsupervised learning*, and *distributed training*. In addition, an effort in integrating the *continual reinforcement learning* paradigm already produced a prototype fork of Avalanche (*avalanche-rl*[1]).

Finally, the *Reproducible Continual Learning* project[2] was created to verify that the Avalanche implementations of popular continual learning techniques are aligned with the results reported in the original papers. Considering that one of the main goals of Avalanche is to enhance the reproducibility of experiments, the RCL effort will be constantly expanded to make sure that the proposed implementations are correct.

6.4 Practical applications

The final category of contributions regards the practical applications described in Chapter 5. In particular, the Continual Object Recognition (CORe) application, which is the first of its kind, was described in Section 5.1. This Android application allows the user to incrementally i) learn to recognize new objects, ii) increase the recognition capabilities on existing classes. The application can complete all training and inference-time tasks offline, without using ad-hoc accelerators. In Section 5.2, the deployment of the AR1* and Latent Replay techniques in an Ultra-Low-Power (ULP) embedded board is described. As of our knowledge, no such extreme deployment scenario has ever been attempted for continual learning techniques.

Research directions In the context of practical vision applications, the natural extension of the aforementioned works is moving towards the detection task. Apart

[1]Avlanche-RL: `https://github.com/ContinualAI/avalanche-rl`
[2]RCL: `https://github.com/ContinualAI/reproducible-continual-learning`

from very few recent works [1], this direction is not much explored in current continual learning literature. However, we argue that classification capabilities may not be enough for the majority of embodied systems.

Appendix A

Details of CWR* and AR1* experiments

This appendix contains the extended information regarding the structure of NICv2 benchmarks not already covered in Section 2.2.2, a list of hyperparameters used to obtain the experimental results found in Section 3.1.2 as well as technical details regarding the reference hardware and software setup. In addition, considering that the CWR* (described in Section 3.1.1) is based on CWR+, the pseudocode for CWR+ is here reported.

A.1 The original CWR+ Strategy

The original CWR+ strategy pseudocode is reported for completeness in Algorithm 5.

The CWR+ strategy was originally targeted at the NC scenario and is not well suited for NI and NIC scenarios due to the fact that previous weights of the CWR layer are overwritten in successive batches (see line 11). This is not an issue in the NC scenario, where each class is encountered in exactly one batch, because class-specific weights are initialized once and are never updated again.

CWR* addresses this issue by updating previously learned weights instead of overwriting them.

Algorithm 5 CWR+ pseudocode for NC scenario where each training batch B_i includes patterns of new classes only: $\bar{\Theta}$ are the class-shared parameters of the representation layers; the notation $cw[j]$ / $tw[j]$ is used to denote the groups of consolidated / temporary weights corresponding to class j. The mean-shift in line 11 allows to adapt the scale of parameters trained in different batches (see Section 3.2 of [103]).

```
1:  procedure CWR+
2:      $cw = 0$
3:      init $\bar{\Theta}$ random or from pre-trained model (e.g. on ImageNet)
4:      for each training batch $B_i$:
5:          expand output layer with neurons for the new classes in $B_i$
6:          $tw = 0$ (for all neurons in the output layer)
7:          train the model with SGD on the $s_i$ classes of $B_i$:
8:              if $B_i = B_1$ learn both $\bar{\Theta}$ and $tw$
9:              else learn $tw$ while keeping $\bar{\Theta}$ fixed
10:         for each class $j$ among the $s_i$ classes in $B_i$
11:             $cw[j] = tw[j] - avg(tw)$
12:         test each class $j$ by using $\bar{\Theta}$ and $cw$
```

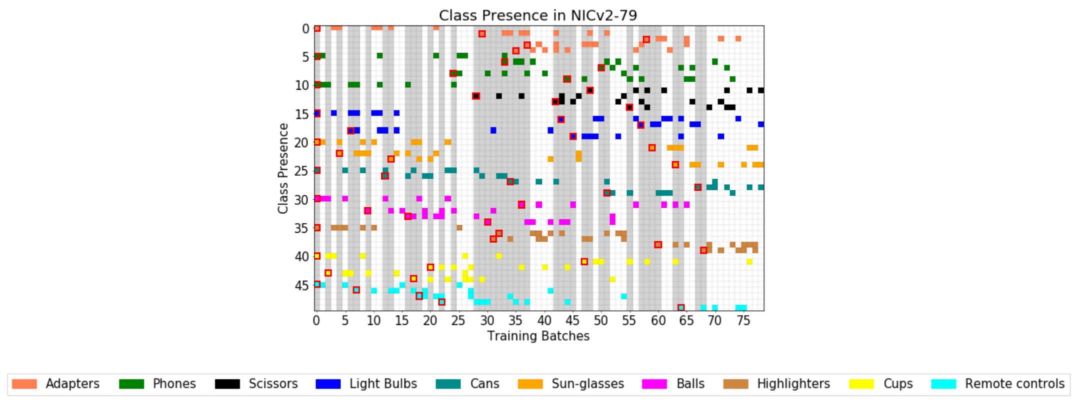

FIGURE A.1: Classes encountered over time in the first run NICv2-79.

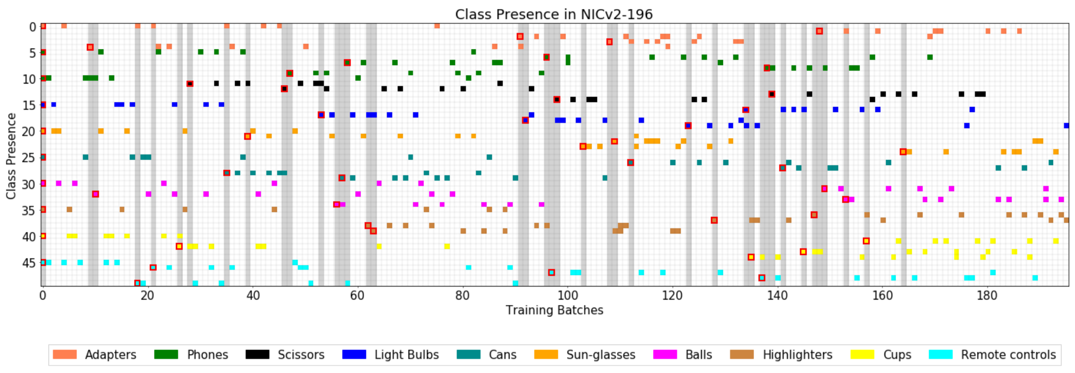

FIGURE A.2: Classes encountered over time in the first run NICv2-196.

A.2 Additional NICv2 visual

Section 2.2.2 introduces the NICv2 benchmarks based on the CORe50 dataset. In particular, Figure 2.4 depicts the NIC and NICv2 benchmarks, both featuring 79 training experiences. In addition, a visual representation of the CORe50-NICv2-391 benchmark is provided in Figure 2.5. The figures here reported (Figure A.1 and A.2) complete the visual description of the benchmarks described in Chapter 2 by providing a representation of the first run of the NICv2-79 (standalone figure) and NICv2-196 benchmarks. In these figures, each row denotes a class. Colors are used to group the 50 classes in the 10 categories. Each column denotes a training batch. A colored block in a (row, column) cell is used to indicate that at least one training session of the row class is present in the column batch. The red-framed cells denote the first introduction of a class. Gray vertical bands highlight experiences where at least one class is seen for the first time.

A.3 Implementation and Experiments Details

For each of the proposed scenarios and strategies, a test accuracy curve was obtained by averaging over 10 different runs. Each run differs from the others by the order of the encountered batches.

All our experiments were executed in a “Ubuntu 16.04” Docker environment using a single GPU. See table A.1 for more details of the host setup. In our experiments we

TABLE A.1: Experimental setup for CWR* and AR1* experiments.

Component	*Model/Version*
Operating System	Debian 8.3
Docker	18.06.1
Nvidia Driver	390.48 (CUDA 9.0, CuDNN 7)
CPU	Intel(R) Xeon(R) CPU E5-2650
GPU	GTX 1080 Ti (11 GB VRAM)
RAM	64 GB DDR3 (1600 MHz)

used a customized version of the Caffe [73] framework. More details can be found in the reference project source code[1].

A.3.1 Hyperparameters

The hyperparameters used in our experiments are described in tables A.2 and A.3. Values reported in table A.3 follow the naming scheme used in [103]. Please note that:

- for AR1* we used two different learning rates: one for the CWR layer and one for the remaining part of the net. This choice can be simply explained by considering that the CWR layer update procedure in AR1* is inherited from the original CWR* strategy. We empirically observed that the overall model performance largely benefits from the use of an higher learning rate for the CWR layer.
- the proposed "Weight Constraining by Learning Rate Modulation" approach was applied to the AR1* strategy only. While this approach could be easily applied to EWC as well, in our experiments we did not change the behavior of the original EWC algorithm. This will allow for a more direct and unbiased comparison of the proposed strategies.

TABLE A.2: Batch ReNormalization parameters.

Parameters	*NICv2-79*	*NICv2-196*	*NICv2-391*
R_{max}	1.25	1.25	1.5
D_{max}	0.5	0.5	2.5
Moving Avg. update rate	0.9999	0.9999	0.9999

[1] https://github.com/lrzpellegrini/Fine-Grained-Continual-Learning

TABLE A.3: Hyperparameter values used in our experiments. The selection was performed on run 0, and hyperparameters were then fixed for runs $1, \ldots, 9$. We used the same set of hyperparameters for the NICv2 79, 196 and 391 scenarios. In this table we use to the same notation introduced in [103].

Naive	
Parameters	*MobileNet V1*
Head	Maximal
B_1: epochs, η (learn. rate)	2, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.000035
LWF	
Parameters	*MobileNet V1*
Head	Maximal
λ	0.1
B_1: epochs, η (learn. rate)	2, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.00005

EWC	
Parameters	*MobileNet V1*
Head	Maximal
max_F	0.001
λ	2.0e6
B_1: epochs, η (learn. rate)	2, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0001
CWR*	
Parameters	*MobileNet V1*
Head	Maximal
B_1: epochs, η (learn. rate)	4, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	4, 0.001
AR1	
Parameters	*MobileNet V1*
Head	Maximal
$w_1, w_i (i > 1)$	0.5, 0.5
max_F	0.001
B_1: epochs, η (learn. rate)	4, 0.001
$B_i, i > 1$: epochs	4
η (learn. rate, CWR layer)	0.001
η (learn. rate,other layers)	0.0001
DSLDA	
Parameters	*MobileNet V1*
shrinkage	1e-4
sigma	plastic

Appendix B

Details of Latent Replay experiments

This appendix contains the useful information regarding the structure of model used to experiment with the latent replay technique. In particular, the architecture and related performance/memory occupation metrics are reported in Section B.1. In addition, the hyperparameters and the hardware/software setup for the experiments found in Section 3.2.3 are reported in B.2 and B.3.

B.1 Model Architecture and Memory Trade-off

In order to assess the trade-off between accuracy, computation, and memory usage we ran AR1* free using different latent replay layers. Table B.1 shows the details of the model we used, which is based on the MobileNetV1 [66], with the only difference that Batch Norm layers have been replaced with Batch ReNorm ones. The network architecture is reported in Table B.1.

TABLE B.1: The architecture of the model used in our experiments. The neurons, weights, and operations are reported for each layer. Those information, along with the results reported in Table 3.2, can be used to identify the most appropriate trade-off between accuracy, computation and used memory depending on the problem context.

Layer	*Neurons*	*Operations*	*Weights*
Images	49152	-	-
conv1	131072	3670016	896
conv2_1/dw	131072	1310720	320
conv2_1/sep	262144	8650752	2112
conv2_2/dw	65536	655360	640
conv2_2/sep	131072	8519680	8320
conv3_1/dw	131072	1310720	1280
conv3_1/sep	131072	16908288	16512
conv3_2/dw	32768	327680	1280
conv3_2/sep	65536	8454144	33024
conv4_1/dw	65536	655360	2560
conv4_1/sep	65536	16842752	65792
conv4_2/dw	16384	163840	2560
conv4_2/sep	32768	8421376	131584
conv5_1/dw	32768	327680	5120
conv5_1/sep	32768	16809984	262656

conv5_2/dw	32768	327680	5120
conv5_2/sep	32768	16809984	262656
conv5_3/dw	32768	327680	5120
conv5_3/sep	32768	16809984	262656
conv5_4/dw	32768	327680	5120
conv5_4/sep	32768	16809984	262656
conv5_5/dw	32768	327680	5120
conv5_5/sep	32768	16809984	262656
conv5_6/dw	8192	81920	5120
conv5_6/sep	16384	8404992	525312
conv6/dw	16384	163840	10240
conv6/sep	16384	16793600	1049600
pool6	1024	16384	0
fc7	50	51250	51250
Total		187,09M	3,35M

B.2 Hyperparameters

The hyperparameters used for the Latent Replay (described in Section 3.2) experiments are reported in tables B.2 and B.3. Please note that:

- The same naming scheme used in [103] is used.
- For AR1* and AR1*free we used a higher learning rate for the CWR layer, as described in Section A.3.1.
- In order to optimize the results for the two different replay types (native rehearsal and latent replay) we chose two different values for the moving average update rate of the BatchReNorm layers [70]. We found out that an higher value of the update rate was better suited for the latent version.
- Excluding the aforementioned update rate, we used the same hyperparameters for both the native and latent rehearsal-based experiments.

TABLE B.2: Hyperparameter values used in our experiments. The selection was performed on run 0, and hyperparameters were then fixed for runs $1, 2, 3, 4$. As an exception for the long running time ($\sim$ 14 days), iCaRL was trained only on run 0.

CWR*	
Parameters	*MobileNet V1*
Head	Maximal
B_1: epochs, η (learn. rate)	4, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	4, 0.003
AR1*	
Parameters	*MobileNet V1*
Head	Maximal
$w_1, w_i(i > 1)$	0.5, 0.5
max_F	0.001
B_1: epochs, η (learn. rate)	4, 0.001
$B_i, i > 1$: epochs	4

η (learn. rate, CWR layer)	0.003
η (learn. rate,other layers)	0.0003

AR1* free

Parameters	*MobileNet V1*
Head	Maximal
B_1: epochs, η (learn. rate)	4, 0.001
$B_i, i > 1$: epochs	4
η (learn. rate, CWR layer)	0.003
η (learn. rate,other layers)	0.0003

DSLDA

Parameters	*MobileNet V1*
Shrinkage	1e-4
Σ	plastic

iCaRL

Parameters	*MobileNet V1*
B_1: epochs, η (learn. rate)	40, 0.01
$B_i, i > 1$: epochs, η (learn. rate)	4, 0.001
K	8000

TABLE B.3: Batch ReNormalization parameters. The reported parameters were used in our experiments on the NICv2-391 scenario involving the CWR*, AR1* and AR1* free algorithms.

Parameters	*Latent Replay*	*Native Rehearsal*
R_{max}	1.25	1.25
D_{max}	0.5	0.5
Moving Avg. update rate	0.99995	0.9999

B.3 Implementation and Experiments Details

For each proposed strategy, a test accuracy curve was obtained by averaging over 5 different runs. The same experimental setup used when experimenting with CWR* and AR1* (described in Section 3.1.2) has been followed, so that each run differs from the others by the order of the encountered experiences. Our experiments were executed in a "Ubuntu 16.04" Docker environment with a custom version of the Caffe [73] framework using a single GPU. See table B.4 for more details of the host setup.

TABLE B.4: Experimental setup for Latent Replay experiments.

Component	*Model/Version*
Operating System	Debian 8.3
Docker	18.06.1
Nvidia Driver	430.40 (CUDA 9.0, CuDNN 7)
CPU	Intel(R) Xeon(R) CPU E5-2650
GPU	GTX 1080 Ti (11 GB VRAM)
RAM	64 GB DDR3 (1600 MHz)

Appendix C

Additional details of the 1st CLVision Challenge

In this appendix, additional details for the solutions submitted at the 1st CLVision Challenge are provided (described in detail in Section 5.4.2). In the following tables, each team and track are reported (see Table C.1, Table C.2, and Table C.2). In particular, we report: i) the model type; ii) if the model was pre-trained; iii) the type of strategy used; iv) the number of eventual replay examples; v) the number of training epochs per batch; vi) the mini-batch size used.

Team	Model	Pre-trained	Strategy	Replay Examples$_{max}$	Epochs	Mini-batch size
UT_LG	DenseNet-161	yes	rehearsal	8000	2	32
Yc14600	ResNeSt50	yes	regularization & rehearsal	12000	1	16
ICT_VIPL	WideResNet-50	yes	rehearsal	4000	2	80
Jodelet	ResNet-50	yes	rehearsal	6400	1	32
Soony	ResNext101/50 & DenseNet161	yes	architectural & rehearsal	119894	1	900
JimiB	resnext101	yes	regularization & rehearsal	11989	8	32
Jun2tong	ResNet-50	yes	regularization & rehearsal	12000	5	32
Sahinyu	Efficientnet-b7	yes	rehearsal	8000	2	27
Ar1	mobilenetV1	yes	architectural	1500	4	128
Noobmaster	resnet-101	yes	rehearsal	24000	5	32
MrGranddy	EfficientNet-B7	yes	regularization & architectural	0	1	32
Naive	mobilenetV1	yes	n.a.	0	4	128
Rehearsal	mobilenetV1	yes	rehearsal	160	4	128
HaoranZhu	ResNet-50	yes	regularization	0	10	32

TABLE C.1: Approaches and baselines details for the NI track.

Team	Model	Pre-trained	Strategy	Replay Examples$_{max}$	Epochs	Mini-batch size
Ar1	mobilenetV1	yes	architectural & rehearsal	1500	4	128
UT_LG	DenseNet-161	yes	architectural	0	1	32
Yc14600	ResNeSt50	yes	regularization & rehearsal	4500	1	16
Soony	ResNext101/50 & DenseNet161	yes	architectural & rehearsal	119890	3	100
Jodelet	ResNet-50	yes	rehearsal	6400	1	32
jun2tong	ResNet-50	yes	regularization & rehearsal	45000	1	32
ICT_VIPL	ResNeXt-50	yes	rehearsal	3000	1	32
Rehearsal	mobilenetV1	yes	rehearsal	180	4	128
JimiB	resnext101	yes	regularization & rehearsal	11989	8	32
MrGranddy	EfficientNet-B7	yes	regularization & architectural	0	1	32
Noobmaster	resnet-101	yes	rehearsal	18000	5	32
HaoranZhu	ResNet-50	yes	regularization	0	10	32
Naive	mobilenetV1	yes	n.a.	0	4	128

TABLE C.2: Approaches and baselines details the MT-NC track.

Team	Model	Pre-trained	Strategy	Replay Examples$_{max}$	Epochs	Mini-batch size
UT_LG	DenseNet-161	yes	rehearsal	78200	1	32
Jodelet	ResNet-50	yes	rehearsal	6400	1	32
Ar1	mobilenetV1	yes	architectural & rehearsal	1500	4	128
ICT_VIPL	ResNet50	yes	rehearsal	2000	1	64
Yc14600	ResNeSt50	yes	regularization & rehearsal	19550	1	32
Rehearsal	mobilenetV1	yes	rehearsal	7820	4	128
Soony	ResNext101/50 & DenseNet161	yes	architectural & rehearsal	119890	1	900
JimiB	resnext101	yes	regularization & rehearsal	11989	6	32
Noobmaster	resnet-101	yes	rehearsal	23460	5	32
Naive	mobilenetV1	yes	n.a.	0	4	128

TABLE C.3: Approaches and baselines details for the NIC track.

Appendix D

Additional details of the IROS 2019 LRV Challenge

This appendix includes additional data regarding the Lifelong Robotic Vision competition held at IROS 2019 (object classification track). In particular, while Section 5.4.1 includes more information regarding the challenge itself and our proposed solution, here additional technical details about other participants' solutions are reported. For a comprehensive description of the challenge, we recommend referring to the final challenge report [9].

D.1 Challenge Method of other participants

Guinness Team The core backend of the approach was the learning without forgetting (LwF) [92]. Figure D.1 illustrates its training strategy. They deployed a pre-trained MobileNet-v2 [149], in which the weights up to the bottleneck are retained as θ_p (θ_p here was fine tuned during training) and they trained the bottleneck weights from scratch. Based on LwF, they retained the θ_{old} that is trained by previous tasks to construct the regularization term for training new weights θ_{new}. It should be noted that there was no replay of previous task images in this structure and only the updated θ_{new} was retained after training. Empirically, they loaded the initial pretrained weights θ_p when processing a new task and θ_p was going to be fine tuned during the training. Details of training scheme are included in Algorithm 6.

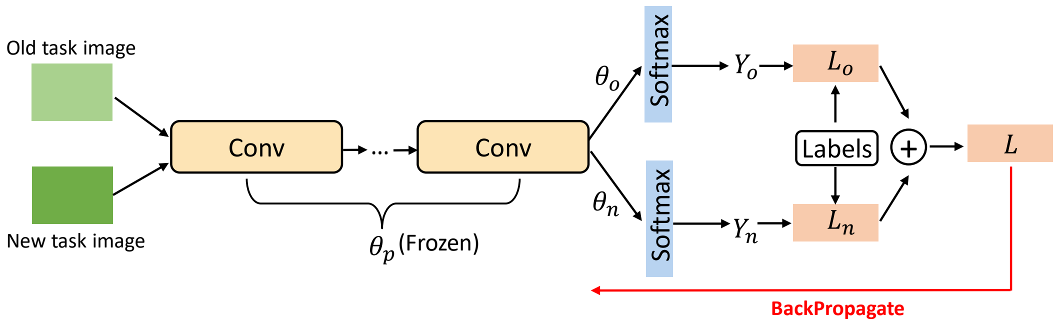

FIGURE D.1: LwF training strategy proposed by the Guinness Team.

```
Algorithm 6 Training details (Guinness Team)
Inputs:
    Training images $\mathbf{X}$, labels $\mathbf{Y}$ of the new task and the pretrained parameters $\theta_p$
Initialize:
    $\mathbf{Y}_o \leftarrow \mathcal{M}_{\hat{\theta}_p,\theta_o}(\mathbf{X})$
    $\theta_n \leftarrow$ Xavier-init($\theta_n$)
    Load the pretrained weights $\theta_p$ to the new model
Train:
    $\theta_p^*, \theta_n^* \leftarrow \underset{\hat{\theta}_p,\hat{\theta}_n}{\text{argmin}}(\lambda\mathcal{L}_o(\mathbf{Y},\mathbf{Y}_o) + \mathcal{L}_n(\mathbf{Y},\mathbf{Y}_n))$
    $\theta_o \leftarrow \theta_n$
```

Neverforget Team The approach was based on Elastic Weight Consolidation (EWC) [79].

As is shown in the Figure D.2, the darker area means a smaller loss or a better solution to the task. First, the parameters of the model are initialized as θ^0 and finetuned as θ^a for Task A. Then, If the model continues to learn Task B and finetuned as θ^b1, the loss of Task A is getting much larger, and it will suffer from the forgetting problem. Instead, the Fisher Information Matrix is utilized to measure the importance of each parameter. If the parameter of the previous task is important, the parameter adjustment in this direction will be constrained and relatively small, if the parameter of the previous task is less important, there will be more space for parameter adjustment in this direction. Assume the importance (the second derivative of log-like function) of parameter θ_2 is more than θ_1, in Task B, the parameter of the neural network will adjust more in θ_1 direction. Thus, the model will gain knowledge of Task B while preserving the knowledge of Task A simultaneously. The ResNet-101 [61] was used as the backbone network. The task was sequentially trained on the training set.

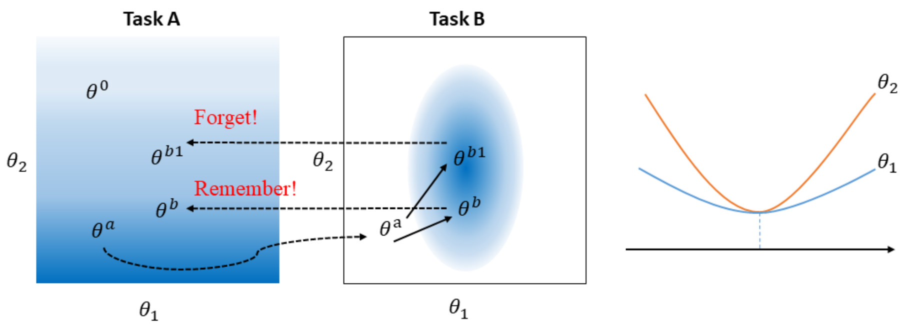

FIGURE D.2: EWC architecture in Neverforget Team's Solution.

SDU_BFA_PKU Team The approach disentangled this problem with two aspects: background removal problem (See Figure D.4) and classification problem.

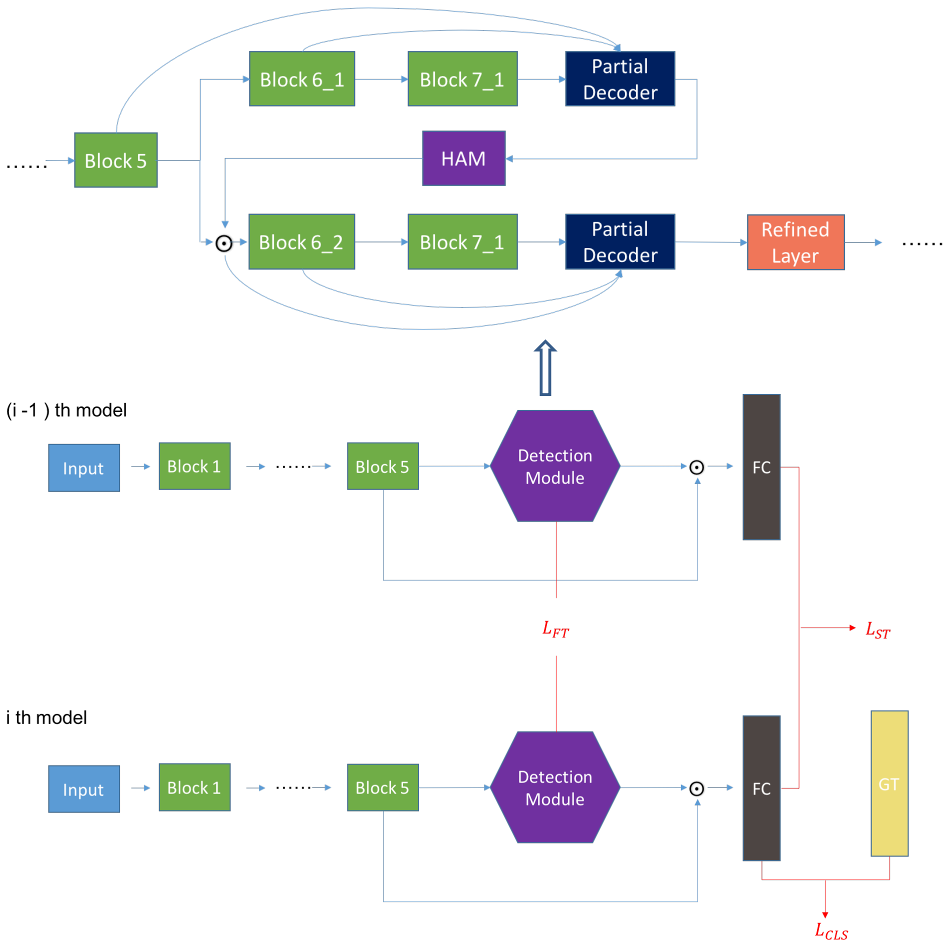

FIGURE D.3: The architecture proposed by the SDU_BFA_PKU Team.

First, they utilized saliency detection method to remove the background noise. Cascaded partial decoder framework which contains two branches is applied to get image saliency map. In each branch, they used a fast and effective partial decoder. The first branch generates an initial saliency map which is utilized to refine the features of the second branch. For classification problem with catastrophic forgetting, they utilized knowledge distillation to prevent it. They used an auto-encoder as a teacher translator, and an encoder as student translator, which has same architecture with teacher translator encoder. The model is aim to project saliency maps from teacher network and student network to same space. Specifically, For i-th task, they regarded $(i-1)$-th model as teacher network, and i-th model as student network. In order to extract the factor from the teacher network, they trained the teacher translator in an unsupervised way by assigning the reconstruction loss at the beginning of every task training process. Then they utilized student translator to translate student network's saliency map output, computed L_1 loss between teacher network output and student network. In order to save computational and storage size, they used MobileNet-v2 as backbone model [149].

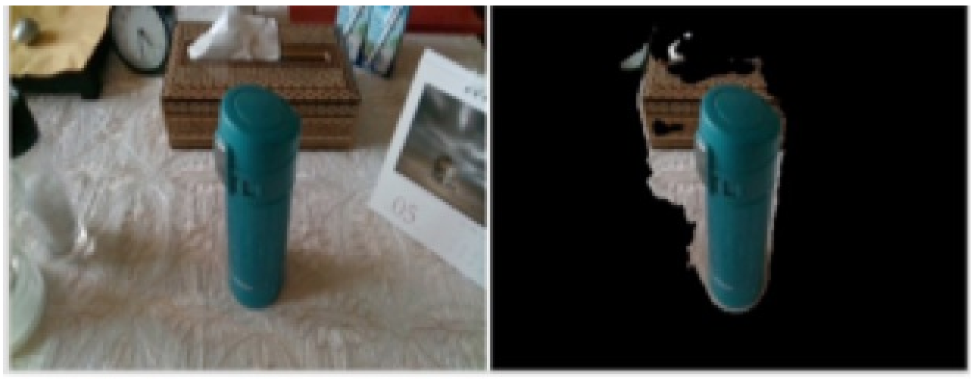

FIGURE D.4: A background removal demo in SDU_BFA_PKU Team's solution.

Vidit98 Team This approach sampled validation data from the buffer and use it as replay data. It intelligently creates the replay memory for a task. Here suppose a network is trained on a task t_n and it learns some feature representation of the images in the task, when trained on the task t_{n+1} it learns the feature representation for images in task t_{n+1}, but as the distribution of data is task t_{n+1} is different, accuracy drops for images in task t_n. The replay memory was an efficient representation of previous tasks data whose information was lost. The replay data was sampled from the validation of all the previous tasks. The network on task t_n is trained and the accuracy of batches of validation data is saved. Next, when trained on task t_i $(i > n)$, the accuracy of same batches of validation data of task t_n is calculated. Then they stored the top k batches from validation data of task t_n whose accuracy has dropped the most. This is done for all the tasks t_0 to t_{i-1}. Training for task t_{i+1} they combined the replay data and training data to train for the particular task. The algorithm is shown in Algorithm 7. The backbone model they used is MobileNet-v2 [149]. Code is made available.

Algorithm 7 Intelligent resampling method (Vidit98 Team)

```
Results: Replay_Data
Initialization:
    $F_i, val\_data_i, t_n, acc[], best\_acc[], topk$
While data in $val\_data_i$ do:
    prec = Accuracy($F_i(data)$)
    Add prec to $acc[]$
end
if $i == n$ then:
    Add $acc$ to $best\_acc$
else
    diff = $best\_acc - acc$
    sort_diff = sort(diff)
    Add $topk$ elements corresponding to sort_diff from $val\_data_i$ to Replay_Data;
```

HYDRA-DI-ETRI Team The team proposed a selective feature learning method to eliminate irrelevant objects in target images. A Single Shot multibox Detection (SSD) algorithm selected desired objects [94]. The SSD algorithm alleviated performance degradation by noisy objects. Then SSD weights were trained with annotated images in task 1, and the refined dataset was fed into a traditional MobileNet [66].

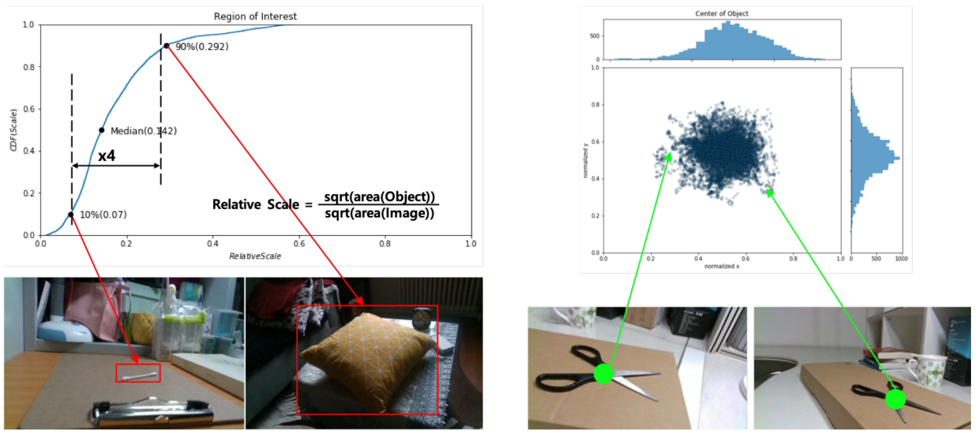

FIGURE D.5: Region of interest analysis in HYDRA-DI-ETRI Team's solution.

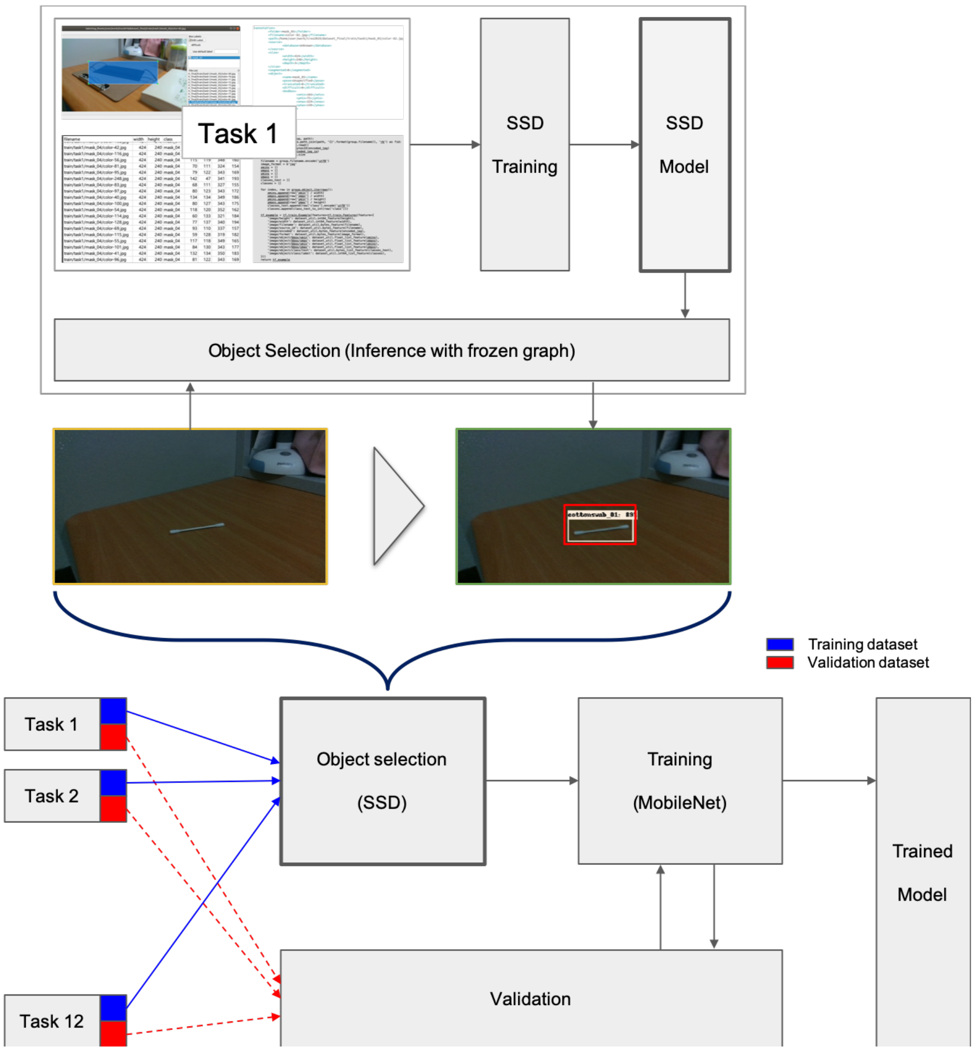

FIGURE D.6: Software architecture for selective feature learning in HYDRA-DI-ETRI Team's solution.

The team also analyzed OpenLORIS-Object dataset to design object recognition software (See Figure D.6), and find that target objects in the dataset coexist with unlabeled objects. The region of interest analysis is illustrated in Figure D.5. Therefore, they proposed a selective feature learning method by eliminating irrelevant features in training dataset. The selective learning procedure is as follows: 1) extracting target objects from training dataset by an object detection algorithm, 2) feeding the refined dataset into a deep neural network to predict labels. In their software, they applied to a SSD as the object detection algorithm due to convenience of flexible feature network design and proper detection performances.

NTU_LL Team The team utilized a combination of Synaptic Intelligence (SI) based regularization method and data augmentation [189] (See Figure D.7). The augmentation strategies they applied were Color Jitter and Blur. ResNet-18 was used for backbone model [61].

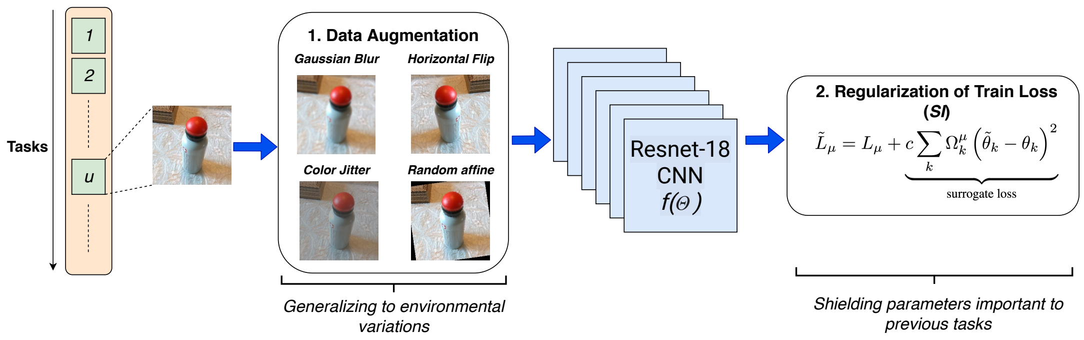
Tasks
1
2
u
1. Data Augmentation
Gaussian Blur
Horizontal Flip
Color Jitter
Random affine
Resnet-18 CNN $f(\Theta)$
2. Regularization of Train Loss (SI)
$\tilde{L}_\mu = L_\mu + c\underbrace{\sum_k \Omega_k^\mu \left(\tilde{\theta}_k - \theta_k\right)^2}_{\text{surrogate loss}}$
Generalizing to environmental variations
Shielding parameters important to previous tasks

FIGURE D.7: Solution architecture proposed by the NTU_LL Team.

D.2 Finalists Information

HIK_ILG Team
Title: Dynamic Neural Network for Incremental Learning
Members: Liang Ma[1], Jianwen Wu[1], Qiaoyong Zhong[1], Di Xie[1] and Shiliang Pu[1]
Affiliation: [1] Hikvision Research Institute, Hangzhou, China.

Unibo Team
Title: Efficient Continual Learning with Latent Rehearsal
Members: Gabriele Graffieti[1], Lorenzo Pellegrini[1], Vincenzo Lomonaco[1] and Davide Maltoni[1]
Affiliation: [1]University of Bologna, Bologna, Italy.

Guinness Team
Title: Learning Without Forgetting Approaches for Lifelong Robotic Vision
Members: Zhengwei Wang[1], Eoin Brophy[2] and Tomás E. Ward[2]
Affiliation: [1]Zhengwei Wang is with V-SENSE, School of Computer Science and Statistics, Trinity College Dublin, Dublin, Irleand; [2]Eoin Brophy and Tomás E. Ward are with the Inisht Centre for Data Analytics, School of Computing, Dublin City University, Dublin, Ireland.

Neverforget Team
Title: A Small Step to Remember: Study of Single Model VS Dynamic Model
Members: Liguang Zhou[1,2]
Affiliation: [1]The Chinese University of Hong Kong (Shenzhen),Shenzhen, China,

[2]Shenzhen Institute of Artificial Intelligence and Robotics for Society, China.

SDU_BFA_PKU Team
Title: SDKD: Saliency Detection with Knowledge Distillation
Members: Lin Yang[1,2,3]
Affiliation: [1]Peking University, Beijing, China, [2]Shandong University, Qingdao, China, [3]Beijing Film Academy, Beijing, China.

Vidit98 Team
Title: Intelligent Replay Sampling for Lifelong Object Recognition
Members: Vidit Goel[1], Debdoot Sheet[1] and Somesh Kumar[1]
Affiliation: [1]Indian Institute of Technology, Kharagpur, India.

HYDRA-DI-ETRI Team
Title: Selective Feature Learning with Filtering Out Noisy Objects in Background Images
Members: Soonyong Song[1], Heechul Bae[1], Hyonyoung Han[1] and Youngsung Son[1]
Affiliation: [1]Electronics and Telecommunications Research Institute (ETRI), Korea.

NTU_LL Team
Title: Lifelong Learning with Regularization and Data Augmentation
Members: Duvindu Piyasena[1], Sathursan Kanagarajah[1], Siew-Kei Lam[1] and Meiqing Wu[1]
Affiliation: [1]Nanyang Technological University, Singapore.

Bibliography

[1] Manoj Acharya, Tyler L. Hayes, and Christopher Kanan. "RODEO: Replay for Online Object Detection". In: *31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020*. BMVA Press, 2020.

[2] Alessandro Achille et al. "Life-Long Disentangled Representation Learning with Cross-Domain Latent Homologies". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 9895–9905.

[3] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. "Task-Free Continual Learning". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[4] Rahaf Aljundi et al. "Gradient Based Sample Selection for Online Continual Learning". In: *Advances in Neural Information Processing Systems 32*. Ed. by H Wallach et al. Curran Associates, Inc., 2019, pp. 11816–11825.

[5] Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. "Safe Policy Search for Lifelong Reinforcement Learning with Sublinear Regret". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 2361–2369.

[6] Haitham Bou Ammar et al. "Autonomous Cross-Domain Knowledge Transfer in Lifelong Policy Gradient Reinforcement Learning". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, pp. 3345–3351. ISBN: 9781577357384.

[7] Haitham Bou Ammar et al. "Online Multi-Task Learning for Policy Gradient Methods". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, pp. II-1206-II–1214.

[8] Bernard Ans and Stéphane Rousset. "Avoiding catastrophic forgetting by coupling two reverberating neural networks". In: *Comptes Rendus de l'Académie des Sciences - Series III - Sciences de la Vie* 320.12 (1997), pp. 989–997. ISSN: 0764-4469.

[9] Heechul Bae et al. "IROS 2019 Lifelong Robotic Vision: Object Recognition Challenge [Competitions]". In: *IEEE Robotics & Automation Magazine* 27.2 (2020), pp. 11–16.

[10] Youngmin Baek et al. "Character Region Awareness for Text Detection". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019, pp. 9357–9366.

[11] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020.

[12] Janusz Bobulski and Mariusz Kubanek. "Deep Learning for Plastic Waste Classification System". In: *Applied Computational Intelligence and Soft Computing* 2021 (May 2021), p. 6626948. ISSN: 1687-9724.

[13] Seok Hoon Boo. *Batch ReNormalization Implementation*. https://github.com/seokhoonboo/caffe-boo. 2017.

[14] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[15] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. "A Systematic Study of the Class Imbalance Problem in Convolutional Neural Networks". In: *Neural Networks* 106 (2018), pp. 249–259.

[16] Alessio Burrello et al. "Work-in-Progress: DORY: Lightweight Memory Hierarchy Management for Deep NN Inference on IoT Endnodes". In: *2019 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2019, pp. 1–2.

[17] Lucas Caccia et al. "Online Learned Continual Compression with Adaptive Quantization Modules". In: (2019).

[18] Massimo Caccia et al. "Online Fast Adaptation and Knowledge Accumulation (OSAKA): a New Approach to Continual Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 16532–16545.

[19] Pablo Samuel Castro et al. *Dopamine: A Research Framework for Deep Reinforcement Learning*. 2018. arXiv: 1812.06110 [cs.LG].

[20] Arslan Chaudhry et al. "Efficient Lifelong Learning with A-GEM". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[21] Arslan Chaudhry et al. *On Tiny Episodic Memories in Continual Learning*. 2019. arXiv: 1902.10486 [cs.LG].

[22] Jiasi Chen and Xukan Ran. "Deep learning with edge computing: A review". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[23] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 1597–1607.

[24] Yu Chen, Tom Diethe, and Peter Flach. *Discriminative Representation Loss (DRL): Connecting Deep Metric Learning to Continual Learning*. 2021. arXiv: 2006.11234 [stat.ML].

[25] Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning*. Morgan & Claypool Publishers, Nov. 2018, pp. 1–187.

[26] Francois Chollet. "Xception: Deep Learning With Depthwise Separable Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[27] Andrea Cossu et al. "Continual Learning for Recurrent Neural Networks: An Empirical Evaluation". In: *Neural Networks* 143 (2021), pp. 607–627. ISSN: 0893-6080.

[28] Matthew Crosby, Benjamin Beyret, and Marta Halina. "The animal-ai olympics". In: *Nature Machine Intelligence* 1.5 (2019), pp. 257–257.

[29] 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)*. Oct. 2021, pp. 8250–8259.

[30] Marc Peter Deisenroth et al. “Multi-task policy search for robotics”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 3876–3881.

[31] Matthias Delange et al. “A continual learning survey: Defying forgetting in classification tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[32] Giorgos Demosthenous and Vassilis Vassiliades. *Continual Learning on the Edge with TensorFlow Lite*. 2021. arXiv: 2105.01946 [cs.LG].

[33] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255.

[34] Prafulla Dhariwal et al. *OpenAI Baselines*. https://github.com/openai/baselines. 2017.

[35] Natalia Díaz-Rodríguez et al. “Don't Forget, There Is More than Forgetting: New Metrics for Continual Learning”. In: *Continual Learning Workshop at NeurIPS* (2018). ISSN: 00954918.

[36] Arthur Douillard and Timothée Lesort. *Continuum: Simple Management of Complex Continual Learning Scenarios*. 2021. arXiv: 2102.06253 [cs.LG].

[37] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey”. In: *J. Mach. Learn. Res.* 20.1 (Jan. 2019), pp. 1997–2017. ISSN: 1532-4435.

[38] Hugo Jair Escalante et al. “AutoML @ NeurIPS 2018 Challenge: Design and Results”. In: *The NeurIPS '18 Competition*. Ed. by Sergio Escalera and Ralf Herbrich. Cham: Springer International Publishing, 2020, pp. 209–229. ISBN: 978-3-030-29135-8.

[39] Falcon, WA et .al. *PyTorch Lightning*. 2019.

[40] Sebastian Farquhar and Yarin Gal. “A Unifying Bayesian View of Continual Learning”. In: *NeurIPS Bayesian Deep Learning Workshop*. 2018.

[41] F. Feng et al. “Challenges in Task Incremental Learning for Assistive Robotics”. In: *IEEE Access* 8 (2020), pp. 3434–3441.

[42] Fernando Fernández and Manuela Veloso. “Learning domain structure through probabilistic policy reuse in reinforcement learning”. In: *Progress in Artificial Intelligence* 2.1 (Mar. 2013), pp. 13–27. ISSN: 2192-6360.

[43] Chrisantha Fernando et al. *PathNet: Evolution Channels Gradient Descent in Super Neural Networks*. 2017. arXiv: 1701.08734 [cs.NE].

[44] Eric Flamand et al. “GAP-8: A RISC-V SoC for AI at the Edge of the IoT”. In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2018, pp. 1–4.

[45] Robert M French. “Catastrophic forgetting in connectionist networks”. In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135. ISSN: 13646613.

[46] Michael Gautschi et al. "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017), pp. 2700–2713.

[47] Georgios Georgiadis. "Accelerating Convolutional Neural Networks via Activation Map Compression". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[48] Alexander Gepperth and Barbara Hammer. "Incremental learning algorithms and applications". In: *European Symposium on Artificial Neural Networks (ESANN)*. Bruges, Belgium, 2016.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[50] Ian J. Goodfellow et al. "An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014.

[51] Google. *Edge TPU*. 2019. URL: http://cloud.google.com/edge-tpu/.

[52] Klaus Greff et al. "The Sacred Infrastructure for Computational Research". In: *Proceedings of the 16th Python in Science Conference*. Ed. by Katy Huff et al. 2017, pp. 49–56.

[53] Stephen T Grossberg. *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control*. Vol. 70. Springer Science & Business Media, 2012.

[54] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 5769–5779. ISBN: 9781510860964.

[55] Raia Hadsell et al. "Embracing Change: Continual Learning in Deep Neural Networks". In: *Trends in Cognitive Sciences* (2020). ISSN: 13646613.

[56] Matthew Hartley and Tjelvar S.G. Olsson. "dtoolAI: Reproducibility for Deep Learning". In: *Patterns* 1.5 (2020), p. 100073. ISSN: 2666-3899.

[57] Tyler L Hayes et al. "New metrics and experimental paradigms for continual learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2031–2034.

[58] Tyler L. Hayes, Nathan D. Cahill, and Christopher Kanan. "Memory Efficient Experience Replay for Streaming Learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 9769–9776.

[59] Tyler L. Hayes and Christopher Kanan. "Lifelong Machine Learning With Deep Streaming Linear Discriminant Analysis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2020.

[60] Tyler L. Hayes et al. "Replay in Deep Learning: Current Approaches and Missing Biological Elements". In: *Neural Computation* 33.11 (Oct. 2021), pp. 2908–2950. ISSN: 0899-7667. eprint: https://direct.mit.edu/neco/article-pdf/33/11/2908/1966599/neco_a_01433.pdf.

[61] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[62] Xin He, Kaiyong Zhao, and Xiaowen Chu. "AutoML: A survey of the state-of-the-art". In: *Knowledge-Based Systems* 212 (2021), p. 106622. ISSN: 0950-7051.

[63] Timm Hess et al. "A Procedural World Generation Framework for Systematic Evaluation of Continual Learning". In: (2021). arXiv: 2106.02585 [cs.LG].

[64] Ashley Hill et al. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018.

[65] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[66] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[67] Jeremy Howard and Sylvain Gugger. "Fastai: A layered API for deep learning". In: *Information* 11.2 (2020), p. 108.

[68] Yen-Chang Hsu et al. *Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines*. 2019. arXiv: 1810.12488 [cs.LG].

[69] Matthew Hutson. *Artificial intelligence faces reproducibility crisis*. 2018.

[70] Sergey Ioffe. "Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1942–1950. ISBN: 9781510860964.

[71] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International Conference on Machine Learning (ICML)* 10.6 (Aug. 2015), pp. 448–456.

[72] C. Jeangoudoux and C. Lauter. "A Correctly Rounded Mixed-Radix Fused-Multiply-Add". In: *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*. 2018, pp. 21–28.

[73] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 675–678. ISBN: 9781450330633.

[74] Matthew Johnson et al. "The Malmo Platform for Artificial Intelligence Experimentation". In: *IJCAI*. 2016.

[75] Heechul Jung et al. "Less-Forgetful Learning for Domain Expansion in Deep Neural Networks". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.

[76] Christoph Käding et al. "Fine-tuning deep neural networks in continuous learning scenarios". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10118 LNCS (2017), pp. 588–605. ISSN: 16113349.

[77] Khimya Khetarpal et al. *Towards Continual Reinforcement Learning: A Review and Perspectives*. 2020. arXiv: 2012.13490 [cs.LG].

[78] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[79] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. ISSN: 0027-8424. eprint: https://www.pnas.org/content/114/13/3521.full.pdf.

[80] Jakub Konečný et al. "Federated Optimization: Distributed Machine Learning for On-Device Intelligence". In: (2016).

[81] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[82] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782.

[83] John K. Kruschke. "ALCOVE: an exemplar-based connectionist model of category learning". In: *Psychological review* 99 1 (1992), pp. 22–44.

[84] Abhishek Kumar and Hal Daume. "Learning Task Grouping and Overlap in Multi-Task Learning". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 1723–1730. ISBN: 9781450312851.

[85] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687.

[86] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[87] Timothée Lesort et al. "Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges". en. In: *Information Fusion* 58 (2020), pp. 52–68. ISSN: 1566-2535.

[88] Timothée Lesort et al. "Generative Models from the perspective of Continual Learning". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. July 2019, pp. 1–8.

[89] Vladimir I Levenshtein et al. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet physics doklady*. Vol. 10. 8. Soviet Union. 1966, pp. 707–710.

[90] Dawei Li et al. "RILOD: Near Real-Time Incremental Learning for Object Detection at the Edge". In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. SEC '19. Arlington, Virginia: Association for Computing Machinery, 2019, pp. 113–126. ISBN: 9781450367332.

[91] Xilai Li et al. "Learn to Grow: A Continual Structure Learning Framework for Overcoming Catastrophic Forgetting". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3925–3934.

[92] Zhizhong Li and Derek Hoiem. "Learning without Forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (Dec. 2018), pp. 2935–2947. ISSN: 1939-3539.

[93] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European Conference on Computer Vision (ECCV)*. Springer. 2014, pp. 740–755.

[94] Wei Liu et al. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[95] Vincenzo Lomonaco and Davide Maltoni. "CORe50: a New Dataset and Benchmark for Continuous Object Recognition". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, Nov. 2017, pp. 17–26.

[96] Vincenzo Lomonaco, Davide Maltoni, and Lorenzo Pellegrini. "Rehearsal-Free Continual Learning over Small Non-I.I.D. Batches". In: *CVPR Workshop on Continual Learning for Computer Vision*. 2020, pp. 246–247.

[97] Vincenzo Lomonaco et al. "Avalanche: An End-to-End Library for Continual Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2021, pp. 3600–3610.

[98] Vincenzo Lomonaco et al. "Continual Reinforcement Learning in 3D Non-Stationary Environments". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 248–249.

[99] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient Episodic Memory for Continual Learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. 2017, pp. 6470–6479. ISBN: 9781510860964.

[100] Zheda Mai et al. *Batch-level Experience Replay with Review for Continual Learning*. 2020. arXiv: 2007.05683 [cs.LG].

[101] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. "Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights". In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari et al. Cham: Springer International Publishing, 2018, pp. 72–88. ISBN: 978-3-030-01225-0.

[102] Arun Mallya and Svetlana Lazebnik. "PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2018, pp. 7765–7773.

[103] Davide Maltoni and Vincenzo Lomonaco. "Continuous learning in single-incremental-task scenarios". In: *Neural Networks* 116 (2019), pp. 56–73. ISSN: 0893-6080.

[104] Davide Maltoni and Vincenzo Lomonaco. "Semi-supervised Tuning from Temporal Coherence". In: *23rd International Conference on Pattern Recognition (ICPR 2016)*. 2016, pp. 2509–2514. ISBN: 978-1-5090-4847-2.

[105] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[106] Marc Masana et al. *Class-incremental learning: survey and performance evaluation on image classification*. 2021. arXiv: 2010.15277 [cs.LG].

[107] Michael McCloskey and Neal J. Cohen. "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem". In: ed. by Gordon H. Bower. Vol. 24. Psychology of Learning and Motivation. Academic Press, 1989, pp. 109–165.

[108] Brendan McMahan et al. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, Apr. 2017, pp. 1273–1282.

[109] Vigneshkumaran Meeradevi Sharavana Raju K. “Automatic Plastic Waste Segregation And Sorting Using Deep Learning Model”. In: *International Journal of Scientific & Technology Research* 9.2 (Feb. 2020), pp. 5773–5777. ISSN: 2277-8616.

[110] Seyed Iman Mirzadeh et al. “Understanding the Role of Training Regimes in Continual Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 7308–7320.

[111] Tom M. Mitchell and Sebastian Thrun. “Explanation-Based Neural Network Learning for Robot Control”. In: *Advances in Neural Information Processing Systems 5, [NIPS Conference]*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 287–294. ISBN: 1558602747.

[112] M. G. Sarwar Murshed et al. “Machine Learning at the Network Edge: A Survey”. In: *ACM Comput. Surv.* 54.8 (Oct. 2021). ISSN: 0360-0300.

[113] Giang Nguyen et al. *ContCap: A scalable framework for continual image captioning*. 2020. arXiv: 1909.08745 [cs.CV].

[114] Fabrice Normandin et al. *Sequoia: A Software Framework to Unify Continual Learning Research*. 2021.

[115] German I. Parisi et al. “Continual lifelong learning with neural networks: A review”. In: *Neural Networks* 113 (May 2019), pp. 54–71. ISSN: 08936080.

[116] German I. Parisi et al. “Lifelong Learning of Spatiotemporal Representations With Dual-Memory Recurrent Self-Organization”. In: *Frontiers in Neurorobotics* 12 (Nov. 2018), pp. 1–20. ISSN: 1662-5218.

[117] Giulia Pasquale et al. “Are we done with object recognition? The iCub robot’s perspective”. In: *Robotics and Autonomous Systems* 112 (2019), pp. 260–281. ISSN: 0921-8890.

[118] Giulia Pasquale et al. *On the Fly Object Recognition on the R1, Your Personal Humanoid - IIT*. https://www.youtube.com/watch?v=HdmDYIL48H4. 2017.

[119] Giulia Pasquale et al. “Teaching iCub to recognize objects using deep Convolutional Neural Networks”. In: *Proceedings of Workshop on Machine Learning for Interactive Systems*. 2015, pp. 21–25.

[120] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[121] Lorenzo Pellegrini et al. “Latent Replay for Real-Time Continual Learning”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2020, pp. 10203–10209.

[122] Lorenzo Pellegrini et al. “Latent replay for real-time continual learning”. In: *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[123] Anastasia Pentina and Christoph H Lampert. "Lifelong Learning with Non-i.i.d. Tasks". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015.

[124] Joelle Pineau et al. "Improving Reproducibility in Machine Learning Research(A Report from the NeurIPS 2019 Reproducibility Program)". In: *Journal of Machine Learning Research* 22.164 (2021), pp. 1–20.

[125] Jary Pomponi et al. "Efficient Continual Learning in Neural Networks with Embedding Regularization". en. In: *Neurocomputing* (2020). ISSN: 0925-2312.

[126] Ameya Prabhu, Philip H. S. Torr, and Puneet K. Dokania. "GDumb: A Simple Approach That Questions Our Progress in Continual Learning". en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 524–540. ISBN: 978-3-030-58536-5.

[127] Antonio Pullini et al. "Mr. wolf: A 1 gflop/s energy-proportional parallel ultra low power soc for iot edge processing". In: *ESSCIRC 2018-IEEE 44th European Solid State Circuits Conference (ESSCIRC)*. IEEE. 2018, pp. 274–277.

[128] Qi She et al. *OpenLORIS Dataset*. `https://lifelong-robotic-vision.github.io/dataset/Data_Object-Recognition.html`. 2019.

[129] Joaquin Quionero-Candela et al. *Dataset shift in machine learning*. The MIT Press, 2009.

[130] Joaquin Quionero-Candela et al. "Pascale 2 challenge: Learning when test and training inputs have different distributions challenge". In: 2005.

[131] Edward Raff. "A Step Toward Quantifying Independently Reproducible Machine Learning Research". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[132] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.

[133] C. Radhakrishna Rao. "The Utilization of Multiple Measurements in Problems of Biological Classification". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 10.2 (1948), pp. 159–193. ISSN: 0035-9246.

[134] Dushyant Rao et al. "Continual Unsupervised Representation Learning". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[135] Sylvestre-Alvise Rebuffi et al. "iCaRL: Incremental Classifier and Representation Learning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[136] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 779–788.

[137] Erik M. Rehn and Davide Maltoni. "Incremental Learning by Message Passing in Hierarchical Temporal Memory". In: *Neural Computation* 26.8 (Aug. 2014), pp. 1763–1809. ISSN: 0899-7667.

[138] Albert Reuther et al. "Survey and Benchmarking of Machine Learning Accelerators". In: *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. Sept. 2019, pp. 1–9.

[139] Michael D Richard and Richard P Lippmann. “Neural Network Classifiers Estimate Bayesian a Posteriori Probabilities”. In: *Neural Computation* 3.4 (1991), pp. 461–483.

[140] Mark Bishop Ring. “Continual Learning in Reinforcement Environments”. UMI Order No. GAX95-06083. PhD thesis. USA, 1994.

[141] Ryne Roady et al. “Stream-51: Streaming classification and novelty detection from videos”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 228–229.

[142] Anthony Robins. “Catastrophic Forgetting, Rehearsal and Pseudorehearsal”. In: *Connection Science* 7.2 (1995), pp. 123–146. ISSN: 13600494.

[143] David Rolnick et al. “Experience Replay for Continual Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[144] Davide Rossi et al. “PULP: A parallel ultra low power platform for next generation IoT applications”. In: *27th IEEE Hot Chips Symposium, HCS 2015*. Institute of Electrical and Electronics Engineers Inc. 2016, pp. 1–39.

[145] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (Apr. 2015), pp. 211–252. ISSN: 1573-1405.

[146] Bryan C Russell et al. “LabelMe: a database and web-based tool for image annotation”. In: *International Journal of Computer Vision* 77.1-3 (2008), pp. 157–173.

[147] Andrei A. Rusu et al. *Progressive Neural Networks*. 2016. arXiv: 1606.04671 [cs.LG].

[148] Paul Ruvolo and Eric Eaton. “ELLA: An Efficient Lifelong Learning Algorithm”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, June 2013, pp. 507–515.

[149] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2018, pp. 4510–4520.

[150] Roy Schwartz et al. “Green AI”. In: *Commun. ACM* 63.12 (Nov. 2020), pp. 54–63. ISSN: 0001-0782.

[151] Jonathan Schwarz et al. “Progress & Compress: A Scalable Framework for Continual Learning”. en. In: *International Conference on Machine Learning*. 2018, pp. 4528–4537.

[152] Joan Serra et al. “Overcoming Catastrophic Forgetting with Hard Attention to the Task”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 4548–4557.

[153] Joan Serra et al. “Overcoming Catastrophic Forgetting with Hard Attention to the Task”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 4548–4557.

[154] Ajmal Shahbaz et al. *International Workshop on Continual Semi-Supervised Learning: Introduction, Benchmarks and Baselines*. 2021.

[155] David Shapiro. *RecyClean*. https://devpost.com/software/resin-id-sorter. 2020.

[156] Qi She and Rosa HM Chan. "Stochastic Dynamical Systems Based Latent Structure Discovery in High-Dimensional Time Series". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 886–890.

[157] Qi She and Anqi Wu. "Neural Dynamics Discovery via Gaussian Process Recurrent Neural Networks". In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. Ed. by Ryan P. Adams and Vibhav Gogate. Vol. 115. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 454–464.

[158] Qi She et al. "OpenLORIS-Object: A Robotic Vision Dataset and Benchmark for Lifelong Deep Learning". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 4767–4773.

[159] Qi She et al. "Reduced-rank linear dynamical systems". In: *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*. 2018.

[160] Hanul Shin et al. "Continual Learning with Deep Generative Replay". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 2994–3003. ISBN: 9781510860964.

[161] Daniel L. Silver and Robert E. Mercer. "The Task Rehearsal Method of Life-Long Learning: Overcoming Impoverished Data". In: *Advances in Artificial Intelligence*. Ed. by Robin Cohen and Bruce Spencer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 90–101. ISBN: 978-3-540-47922-2.

[162] Daniel L. Silver and Ryan Poirier. "Sequential Consolidation of Learned Task Knowledge". In: *Advances in Artificial Intelligence*. Ed. by Ahmed Y. Tawfik and Scott D. Goodwin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 217–232. ISBN: 978-3-540-24840-8.

[163] R. Smith. "An Overview of the Tesseract OCR Engine". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. 2007, pp. 629–633.

[164] STM32 Stmicroelectronics. *stm32h743 datasheet*. 2018.

[165] Niko Sünderhauf et al. "The limits and potentials of deep learning for robotics". In: *The International Journal of Robotics Research* 37.4-5 (Apr. 2018), pp. 405–420. ISSN: 0278-3649.

[166] Mingxing Tan and Quoc Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 6105–6114.

[167] Fumihide Tanaka and Masayuki Yamamura. "An approach to lifelong reinforcement learning through multiple environments". In: *6th European Workshop on Learning Robots*. 1997, pp. 93–99.

[168] Norman Tasfi. *PyGame Learning Environment*. https://github.com/ntasfi/PyGame-Learning-Environment. 2016.

[169] Anh Thai et al. "Does Continual Learning = Catastrophic Forgetting?" In: *arXiv* (2021). arXiv: 2101.07295.

[170] Neil C. Thompson et al. *The Computational Limits of Deep Learning*. 2020. arXiv: 2007.05558 [cs.LG].

[171] Sebastian Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. USA: Kluwer Academic Publishers, 1996. ISBN: 0792397169.

[172] Sebastian Thrun and Tom M. Mitchell. "Lifelong robot learning". In: *Robotics and Autonomous Systems* 15.1 (1995). The Biology and Technology of Intelligent Autonomous Agents, pp. 25–46. ISSN: 0921-8890.

[173] Alan M. Turing. "Computing Machinery and Intelligence". In: *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*. Ed. by Robert Epstein, Gary Roberts, and Grace Beber. Dordrecht: Springer Netherlands, 2009, pp. 23–65. ISBN: 978-1-4020-6710-5.

[174] Anastasiia Usmanova et al. "A distillation-based approach integrating continual learning and federated learning for pervasive services". In: (2021).

[175] Gido M. van de Ven, Hava T. Siegelmann, and Andreas S. Tolias. "Brain-inspired replay for continual learning with artificial neural networks". In: *Nature Communications* 11.1 (Aug. 2020), p. 4069. ISSN: 2041-1723.

[176] Gido M. van de Ven and Andreas S. Tolias. *Generative replay with feedback connections as a general strategy for continual learning*. 2019. arXiv: 1809.10635 [cs.LG].

[177] Gido M. van de Ven and Andreas S. Tolias. *Three scenarios for continual learning*. 2019. arXiv: 1904.07734 [cs.LG].

[178] Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. "Efficient Continual Learning with Modular Networks and Task-Driven Priors". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[179] Liyuan Wang et al. "ORDisCo: Effective and Efficient Usage of Incremental Unlabeled Data for Semi-Supervised Continual Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 5383–5392.

[180] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. "Growing a Brain: Fine-Tuning by Increasing Model Capacity". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 3029–3038.

[181] Juyang Weng et al. "Autonomous Mental Development by Robots and Animals". In: *Science* 291.5504 (2001), pp. 599–600. eprint: https://www.science.org/doi/pdf/10.1126/science.291.5504.599.

[182] Aaron Wilson et al. "Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach". In: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, pp. 1015–1022. ISBN: 9781595937933.

[183] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[184] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. "ViZDoom Competitions: Playing Doom from Pixels". In: *IEEE Transactions on Games* (2018).

[185] Zichen Xu, Li Li, and Wenting Zou. "Exploring federated learning on battery-powered devices". In: *Proceedings of the ACM Turing Celebration Conference-China*. 2019, pp. 1–6.

[186] I. Zeki Yalniz et al. *Billion-scale semi-supervised learning for image classification*. 2019. arXiv: `1905.00546 [cs.CV]`.

[187] Mindy Yang and Gary Thung. "Classification of trash for recyclability status". In: *CS229 Project Report* 2016 (2016).

[188] Jaehong Yoon et al. "Lifelong Learning with Dynamically Expandable Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[189] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual Learning Through Synaptic Intelligence". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. 2017, pp. 3987–3995.

[190] Zhi Zhou et al. "Edge intelligence: Paving the last mile of artificial intelligence with edge computing". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1738–1762.