Avalanche: an End-to-End Library for Continual Learning

Vincenzo Lomonaco[1†*] Lorenzo Pellegrini[2†] Andrea Cossu[1,18†] Antonio Carta[1†] Gabriele Graffieti[2†]
Tyler L. Hayes[3] Matthias De Lange[4] Marc Masana[5] Jary Pomponi[6] Gido van de Ven[7]
Martin Mundt[8] Qi She[9] Keiland Cooper[10] Jeremy Forest[11] Eden Belouadah[12]
Simone Calderara[13] German I. Parisi[14] Fabio Cuzzolin[15] Andreas Tolias[7] Simone Scardapane[6]
Luca Antiga[16] Subutai Amhad[17] Adrian Popescu[12] Christopher Kanan[3]
Joost van de Weijer[5] Tinne Tuytelaars[4] Davide Bacciu[1] Davide Maltoni[2]

[1]University of Pisa [2]University of Bologna [3]Rochester Institute of Technology
[4]KU Leuven [5]Universitat Autònoma de Barcelona [6]Sapienza University of Rome
[7]Baylor College of Medicine [8]Goethe University [9]ByteDance AI Lab
[10]University of California [11]New York University [12]Université Paris-Saclay
[13]University of Modena and Reggio-Emilia [14]University of Hamburg
[15]Oxford Brookes University [16]Orobix [17]Numenta [18]Scuola Normale Superiore

https://avalanche.continualai.org

Abstract

Learning continually from non-stationary data streams is a long-standing goal and a challenging problem in machine learning. Recently, we have witnessed a renewed and fast-growing interest in continual learning, especially within the deep learning community. However, algorithmic solutions are often difficult to re-implement, evaluate and port across different settings, where even results on standard benchmarks are hard to reproduce. In this work, we propose Avalanche, an open-source end-to-end library for continual learning research based on PyTorch. Avalanche is designed to provide a shared and collaborative codebase for fast prototyping, training, and reproducible evaluation of continual learning algorithms.

1. Introduction

Continual Learning (CL), also referred to as *Lifelong* or *Incremental Learning*, is a challenging research problem [7]. Lately, it has become the object of fast-growing interest from the research community, especially thanks to recent investigations leveraging gradient-based deep architectures [15, 37]. In the last few years, machine learning has witnessed a prolific, variegated, and original research production on the topic: from Computer Vision [9, 30, 36] to Robotics [28, 40, 38], from Reinforcement Learning [25, 31] to Sequence Learning [8], among others.

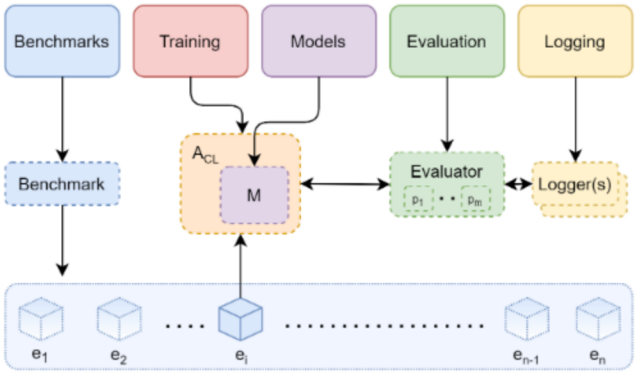


Figure 1: Operational representation of Avalanche with its main modules (top), the main object instances (middle) and the generated stream of data (bottom). A *Benchmark* generates a stream of experiences e_i which are sequentially accessible by the continual learning algorithm A_{CL} with its internal 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.

However, continual learning algorithms today are often designed and implemented from scratch with different assumptions, settings, and benchmarks that make them diffi-

*Corresponding author: vincenzo.lomonaco@unipi.it
†Avalanche lead maintainers.

cult to compare among each other or even port to slightly different contexts. A crucial factor for the consolidation of a fast-growing research topic in the machine learning domain is the availability of tools and libraries easing the implementation, assessment, and replication of models across different settings, while promoting the reproducibility of results from the literature [41].

In this work, we propose *Avalanche*, an open-source (MIT licensed) end-to-end library for continual learning based on PyTorch [39], devised to provide a shared and collaborative codebase for fast prototyping, training, and evaluation of continual learning algorithms.

We designed Avalanche according to a set of fundamental design principles (Sec. 2) which, we believe, can help researchers and practitioners in a number of ways: i) *Write less code, prototype faster, and reduce errors*; ii) *Improve reproducibility*; iii) *Favor modularity and reusability*; iv) *Increase code efficiency, scalability and portability*; v) *Foster impact and usability*.

The contributions of this paper can be summarized as follows:

1. We propose a general continual learning framework that provides the conceptual foundation for Avalanche (Sec. 3).
2. We discuss the general design of the library based on five main modules: *Benchmarks*, *Training*, *Evaluation*, *Models*, and *Logging* (Sec. 4).
3. We release the open-source, collaboratively maintained project at `https://github.com/ContinualAI/avalanche`, as the result of a collaboration involving over 15 organizations across Europe, United States, and China.

Prior work related to this project is discussed in Section 5. Lastly, we discuss in Section 6 the importance that an *all-in-one*, *community-driven* library may have for the future of continual learning and its possible extensions.

2. Design Principles

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

Comprehensiveness and consistency 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.

Ease-of-Use The second principle is the focus on *simplicity*: simple solutions to complex problems and a simple usage of the library. We concentrate 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 [21]. 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.

Modularity and Independence *Modularity* is another fundamental design principle. 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. A particular focus on module *independence* is maintained to guarantee the stand-alone usability of individual module functionalities and facilitates learning of a particular tool.

Efficiency and Scalability Computational and memory requirements in machine learning have grown significantly throughout the last two decades [52]. Standard deep learning libraries such as TensorFlow [1] or PyTorch [39] already focus on *efficiency* and *scalability* as two fundamental designing principles, since modern research experiments can take months to complete [48]. 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.

3. Continual Learning Framework

Recently, we have witnessed a significant attempt to formalize a general framework for continual learning algorithms [30, 31, 54]. These proposals often categorize scenarios and algorithms based on their unique properties and

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

specific settings. However, as outlined in this paper, within the formal design of Avalanche, we take a different approach.

Given the fast-evolving, often conflicting views of the problem, we aim 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*:

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

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. Main Modules

The library is organized into five main modules: **Benchmarks** (Sec. 4.1), **Training** (Sec. 4.2), **Evaluation** (Sec. 4.3), **Models** (Sec. 4.5), and **Logging** (Sec. 4.4). Table 1 summarizes the features provided by Avalanche at the current stage of development. In Fig. 1, an operational representation of the library modules and their interplay within the aforementioned reference framework is shown.

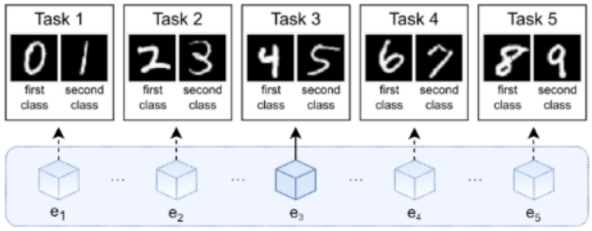


Figure 2: Example of a generated stream in Avalanche, composed by five experiences, implementing the common SplitMNIST benchmark [57]. When accessing experience e_3, the A_{CL} algorithm has no access to previous or future experiences.

4.1. Benchmarks

Continual learning revolves around the idea of dealing with a non-stationary stream of experiences. An example stream from the standard SplitMNIST benchmark [57] composed of five experiences is shown in Fig. 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 [36]) 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 and boundaries [2], etc. When defining such elements, some degree of freedom is retained to allow obtaining different *benchmark instances*. For example, different instantiations of the SplitMNIST benchmark [57] can be obtained by setting different class assignments. Alternatively, distinct instances of the PermutedMNIST [14] benchmark can be obtained by choosing different pixel permutations.

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 [45], PermutedMNIST [14], etc.) through the *classic* submodule. A simple example illustrating how to use the "SplitMNIST" benchmark [57] is shown in Fig. 3. Moreover, a wide range of tools are available that enables the creation of customized benchmarks. 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

	Supported features
Benchmarks	Split/Permuted/Rotated MNIST [34], Split Fashion Mnist [13], Split Cifar10/100/110 [45, 35], Split CUB200, Split ImageNet [45], Split TinyImageNet [9], Split/Permuted/Rotated Omniglot [49], CORe50 [32], OpenLORIS [51], Stream51 [47].
Scenarios	Multi Task [28], Single Incremental Task [28], Multi Incremental Task [28], Class incremental [46, 54], Domain Incremental [54], Task Incremental [54], Task-agnostic, Online, New Instances, New Classes, New Instances and Classes.
Strategies	Naive (Finetuning), CWR* [33], Replay, GDumb [43], Cumulative, LwF [29], GEM [34], A-GEM [6], EWC [26], Synaptic Intelligence [57], AR1 [35], Joint Training.
Metrics	Accuracy, Loss (user specified), Confusion Matrix, Forgetting, CPU Usage, GPU usage, RAM usage, Disk Usage, Timing, Multiply, and Accumulate [22, 11].
Loggers	Text Logger, Interactive Logger, Tensorboard Logger [1], Weights and Biases (W&B) Logger [3] (in progress).

Table 1: Avalanche supported features for the Alpha release (v0.0.1).

Classic Benchmarks

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

Figure 3: Simple instantiation of a *Classic* continual learning benchmark.

dataset implementation provided by the `torchvision` library. A proper implementation is provided for other datasets (such as CORe50 [32], Stream-51 [47], and OpenLORIS [51]). The benchmark preparation and data loading process can seamlessly handle memory-intensive benchmarks, such as Split-ImageNet [45], 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.

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 [35], 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 [54]. 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 [54]. Most classic benchmarks are based on these generators. Fig. 4 shows a simple example of using `nc_scenario`.

Benchmarks Generators

```
# 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: Example of 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 [23], 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 contemplate an out-of-distribution stream, a validation stream and possibly several other arbitrary streams, each linked to a different semantic. For instance, [6] proposes a benchmark where a separate stream of data is used for cross-validation, while [47] 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 properly describe a testing 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 patterns related to classes in the corresponding training experience.

Avalanche currently supports two different streams: *train* and *test*, while the support for arbitrary streams (for instance, out-of-distribution stream) will be implemented in the near future.

Each experience can be obtained by iterating over one of the available streams. Fig. 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.

Task Labels and Nomenclature Every mechanism, internal aspect, name of function and class in the benchmarks

Main Training Loop

```
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 5: Example of the main training loop over the stream of experiences.

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.

4.2. 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 11 continual learning strategies (with many more to come), that can be used to train baselines in a few lines of code, as shown in Fig. 6. See Table 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 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 Fig. 7 shows part of the `BaseStrategy.train` loop (`eval` has a similar structure).

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

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 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) [26] and replay using the snippet of code shown in Fig. 8.

Hybrid Strategies

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

Figure 8: Example of an on-the-fly instantiation of hybrid strategies through Plugins.

4.3. Evaluation

The performance of a CL algorithm should be assessed by monitoring multiple aspects of the computation [28]. 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 4.4). 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 [22] (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. Fig. 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 [11].

4.4. 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 three different loggers: `TextLogger`, `InteractiveLogger` and `TensorboardLogger` [1]. They provide reports on textual file, standard output and Tensorboard, respectively. As soon as a metric emits a value, the Text Logger prints the complete metric name followed by its value in the destination file. See Fig. 10 for an example of Tensorboard output. 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` is able to show images and more complex outputs, which cannot be appropriately printed on standard output or textual file. We are also working on the integration of the Weights and Biases logger [3], which should be released soon.
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 Fig. 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`.

Evaluation Plugin

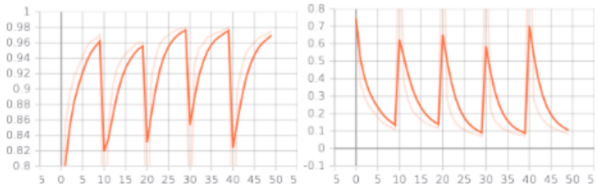
```
text_logger = TextLogger(
    open('out.txt', 'w'))
interactive_logger = InteractiveLogger()
tensorboard_logger = TensorboardLogger()

eval_plugin = EvaluationPlugin(
    accuracy_metrics(experience=True),
    loss_metrics(minibatch=True,
        epoch=True, stream=True),
    ExperienceForgetting(),
    StreamConfusionMatrix(),
    cpu_usage_metrics(stream=True),
    timing_metrics(minibatch=True),
    ram_usage_metrics(epoch=True),
    gpu_usage_metrics(gpu_id, epoch=True),
    disk_usage_metrics(epoch=True),
    MAC_metrics(minibatch=True),

    loggers=[interactive_logger,
             text_logger,
             tensorboard_logger])
```

Figure 9: Avalanche evaluation plugin (or *evaluator*) object instantiation example.

4.5. Models

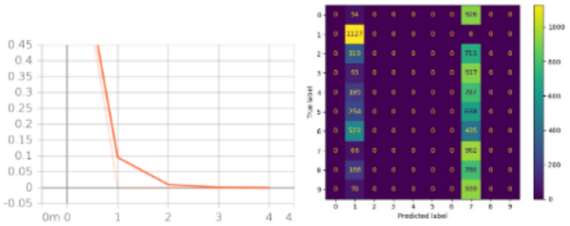
The Avalanche `models` module offers a set of simple machine learning architectures ready to be used in experiments. In particular, the module contains versions of feedforward and convolutional neural networks and a pretrained version of MobileNet (v1) [18]. The main purpose of these models is to let the user focus on Avalanche features, rather than on writing lines of code to build a specific architecture. We plan to extend our model support with more advanced architectures, also tailored to specific CL

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



(a) Accuracy over epochs during training

(b) Loss over epochs during training



(c) Accuracy over first experience across 5 training experiences

(d) Confusion Matrix

Figure 10: `Tensorboard Logger` output examples.

applications.

5. Related Works

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 [16, 42, 44].

The advent of machine (and deep) learning libraries, mainly TensorFlow [1] and PyTorch [39] has partially mitigated the reproducibility problem. Using these libraries assures a standard implementation of many machine learning building blocks, reducing ambiguities due to bespoke and different implementations of basic concepts.

In recent times, the continual learning community has put a lot of effort into addressing these problems, by providing code and libraries aimed to increase the reproducibility of continual learning experiments [9, 20, 36, 50, 53, 54]. On the one hand, these first attempts lack the generality and the consistency of Avalanche, especially regarding the creation of different and complex benchmarks, and the continual support of a large community. On the other hand, they demonstrate, however, the growing interest of the entire community towards these issues.

Another area other than continual learning which 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 [4], within which a multitude of different RL environments is available. A similar library is ViZDoom [56], in which an agent plays the famous computer game *Doom*. Other relevant projects in the field of reinforcement learning are Dopamine [5], which focuses on simplicity and easy prototyping, and project Malmo [24], which is based on the famous Minecraft game.

Many of these libraries, however, only focus on the agent's interaction with the environment (which, in the continual learning domain, can be translated into the definition of benchmarks and scenarios), providing none or just a few base strategies or baselines. In the reinforcement learning field, this problem is addressed by other libraries that include standard implementations of baselines algorithms, such as OpenAI baselines [10] and stable baselines [17].

Another prominent example of a collection of baseline training strategies and pretrained models is the natural language processing transformers library by Hugging Face [55]. 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 [12] and fastai [19]. Indeed, Avalanche is based on the same comprehensiveness and consistency principle, hence not only benchmarks but also strategies and metrics are included. This promotes consistency across the different parts of the library and simplifies the interaction between different modules. Moreover, Avalanche could be integrated with the mentioned deep learning libraries, thanks to the similarity regarding design principles and code structure.

Another important problem in research is the bookkeeping of experiments. Having a tool that keeps track of any single run (hyper-parameters, model used, algorithm used, variants, inputs to the model, etc.) is crucial for reproducibility, especially when strategies such as grid search are applied. As discussed in Sec. 4.4, 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 [27] or Weights and Biases (wandb) [3].

6. Conclusion and Future Work

In the last decade, we have witnessed a significant effort towards making research in machine learning more transparent, reproducible and open-access. However, although research papers are increasingly accompanied by publicly hosted codebases, it is often difficult to run and integrate such software into environments which are typically different from the one within which it was originally designed. This not only hampers reproducibility but also inhibits scal-

ability, for each research paper ends up creating its own implementation almost from scratch.

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 1, the current Avalanche `Alpha` 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 [15]. However, being Avalanche a *community-driven* effort, we plan in both the near and medium terms to support the integration of additional learning paradigms (e.g. Reinforcement or Unsupervised Learning), tasks type (e.g. Detection, Segmentation) and application contexts (e.g. Natural Language Processing, Speech Recognition), depending on the research community demands and priorities.

We hope that this library, as a powerful avalanche, may trigger a positive reinforcement loop within our community, nudging it to shift towards a more collaborative and inclusive research environment and helping all of us tackle together the grand research challenges presented by a frontier topic such as continual learning.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 2, 4, 7, 8

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

[3] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com. 4, 7, 8

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 8

[5] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. *arXiv preprint arXiv:1812.06110*, 2018. 8

[6] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient Lifelong Learning with A-GEM. In *ICLR*, 2019. 4, 5

[7] Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning, Second Edition*, volume 12. Morgan & Claypool Publishers LLC, 2018. 1

[8] Andrea Cossu, Antonio Carta, Vincenzo Lomonaco, and Davide Bacciu. Continual learning for recurrent neural networks: a review and empirical evaluation. *arXiv preprint arXiv:2103.07492*, 2021. 1

[9] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 1, 4, 8

[10] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017. 8

[11] Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. Don't forget, there is more than forgetting: New metrics for Continual Learning. *arXiv*, 2018. 4, 7

[12] WA Falcon and .al. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019. 8

[13] Sebastian Farquhar and Yarin Gal. A Unifying Bayesian View of Continual Learning. In *NeurIPS Bayesian Deep Learning Workshop*, 2018. 4

[14] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013. 3

[15] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing Change: Continual Learning in Deep Neural Networks. *Trends in Cognitive Sciences*, 2020. 1, 9

[16] Matthew Hartley and Tjelvar S.G. Olsson. dtoolai: Reproducibility for deep learning. *Patterns*, 1(5):100073, 2020. 8

[17] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018. 8

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, 2017. 7

[19] Jeremy Howard and Sylvain Gugger. Fastai: A layered api for deep learning. *Information*, 11(2):108, 2020. 8

[20] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In *NeurIPS Continual learning Workshop*, 2018. 8

[21] Matthew Hutson. Artificial intelligence faces reproducibility crisis, 2018. 2

[22] C. Jeangoudoux and C. Lauter. A Correctly Rounded Mixed-Radix Fused-Multiply-Add. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 21–28, 2018. 4, 6

[23] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014. 5

[24] Matthew Johnson, Katja Hofmann, T. Hutton, and D. Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 2016. 8

[25] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020. 1

[26] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *PNAS*, 114(13):3521–3526, 2017. 4, 6

[27] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber. The Sacred Infrastructure for Computational Research. In Katy Huff, David Lippa, Dillon Niederhut, and M Pacer, editors, *Proceedings of the 16th Python in Science Conference*, pages 49 – 56, 2017. 8

[28] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020. 1, 4, 6

[29] Zhizhong Li and Derek Hoiem. Learning without Forgetting. In *European Conference on Computer Vision*, Springer, pages 614–629, 2016. 4

[30] Vincenzo Lomonaco. *Continual Learning with Deep Architectures*. PhD Thesis, alma, 2019. 1, 2

[31] Vincenzo Lomonaco, Karan Desai, Eugenio Culurciello, and Davide Maltoni. Continual Reinforcement Learning in 3D Non-Stationary Environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 248–249, 2020. 1, 2

[32] Vincenzo Lomonaco and Davide Maltoni. CORe50: A New Dataset and Benchmark for Continuous Object Recognition. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 17–26. PMLR, 2017. 4

[33] 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*, pages 246–247, 2020. 4

[34] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. In *NIPS*, 2017. 4

[35] Davide Maltoni and Vincenzo Lomonaco. Continuous Learning in Single-Incremental-Task Scenarios. *Neural Networks*, 116:56–73, 2019. 4

[36] Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation. *arXiv preprint arXiv:2010.15277*, 2020. 1, 3, 8

[37] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. 1

[38] German I Parisi and Vincenzo Lomonaco. Online continual learning on sequences. In *Recent Trends in Learning From Data*, pages 197–221. Springer, 2020. 1

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019. 2, 8

[40] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. 1

[41] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *arXiv preprint arXiv:2003.12206*, 2020. 2

[42] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, V. Larivière, A. Beygelzimer, Florence d'Alché Buc, Emily Fox, and H. Larochelle. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *ArXiv*, abs/2003.12206, 2020. 8

[43] Ameya Prabhu, Philip H. S. Torr, and Puneet K. Dokania. GDumb: A Simple Approach that Questions Our Progress in Continual Learning. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 524–540, Cham, 2020. Springer International Publishing. 4

[44] Edward Raff. A step toward quantifying independently reproducible machine learning research. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 8

[45] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental Classifier and Representation Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 4

[46] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 4

[47] Ryne Roady, Tyler L Hayes, Hitesh Vaidya, and Christopher Kanan. Stream-51: Streaming classification and novelty detection from videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 228–229, 2020. 4, 5

[48] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *arXiv preprint arXiv:1907.10597*, 2019. 2

[49] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & Compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537, 2018. 4

[50] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4548–4557, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 8

[51] Qi She, Fan Feng, Xinyue Hao, Qihan Yang, Chuanlin Lan, Vincenzo Lomonaco, Xuesong Shi, Zhengwei Wang, Yao Guo, Yimin Zhang, Fei Qiao, and Rosa H M Chan. OpenLORIS-Object: A Robotic Vision Dataset and Benchmark for Lifelong Deep Learning. *arXiv*, pages 1–8, 2019. 4

[52] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020. 2

[53] Gido M van de Ven and Andreas S Tolias. Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*, 2018. 8

[54] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. In *Continual Learning Workshop NeurIPS*, 2018. 2, 4, 8

[55] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. 8

[56] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018. 8

[57] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017. 3, 4